SPECTRUM®

Extensions Integration Toolkit
Developer's Guide

CABLETRON SYSTEMS

The Complete Networking Solution™

Summary of Changes

Version	Date	Reason/Rational	Nature of Changes
E12	Sepember, 1997	Release of Beta SPECTRUM 5.0	Removed references to platforms not supported by new software release; redefined dropped sections as “Retired”; removed sections pertaining to magnetic-tape distribution and added procedure for shipping extensions via vtape; made minor style changes throughout.
E13	March 1998	Update for Beta 5.0r3	Replaced references to previous core documentarion with references to new documents replacing those out-of-print manuals; removed Getting Help table, which is now maintained as a part of the CD-ROM Roadmap.

Notice

Cabletron Systems reserves the right to make changes in specifications and other information contained in this document without prior notice. The reader should in all cases consult Cabletron Systems to determine whether any such changes have been made.

The hardware, firmware, or software described in this manual is subject to change without notice.

IN NO EVENT SHALL CABLETRON SYSTEMS BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS MANUAL OR THE INFORMATION CONTAINED IN IT, EVEN IF CABLETRON SYSTEMS HAS BEEN ADVISED OF, KNOWN, OR SHOULD HAVE KNOWN, THE POSSIBILITY OF SUCH DAMAGES.

Virus Disclaimer

Cabletron has tested its software with current virus checking technologies. However, because no anti-virus system is 100% reliable, we strongly caution you to write protect and then verify that the Licensed Software, prior to installing it, is virus-free with an anti-virus system in which you have confidence.

Cabletron Systems makes no representations or warranties to the effect that the Licensed Software is virus-free.

Copyright © March, 1998, by Cabletron Systems, Inc. All rights reserved.

Printed in the United States of America.

Order Number: 9030623 E13

Cabletron Systems, Inc.
P.O. Box 5005
Rochester, NH 03866-5005

SPECTRUM, **SPECTRUM IMT/VNM** logo, **DCM**, **IMT** and **VNM** are registered trademarks, and **SpectroGRAPH**, **SpectroSERVER**, **Device Communications Manager, Inductive Modeling Technology**, **Device Communications Manager**, and **Virtual Network Machine** are trademarks of Cabletron Systems, Inc.

UNIX and **X Window System** are trademarks of The Open Group.
OSF/Motif and **Motif** are trademarks of the Open Software Foundation, Inc.
Ethernet is a trademark of Xerox Corporation.

Restricted Rights Notice

(Applicable to licenses to the United States Government only.)

1. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c) (1) (ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

   Cabletron Systems, Inc., 35 Industrial Way, Rochester, New Hampshire 03866-5005.

2. (a) This computer software is submitted with restricted rights. It may not be used, reproduced, or disclosed by the Government except as provided in paragraph (b) of this Notice or as otherwise expressly stated in the contract.

   (b) This computer software may be:

   (1) Used or copied for use in or with the computer or computers for which it was acquired, including use at any Government installation to which such computer or computers may be transferred;

   (2) Used or copied for use in a backup computer if any computer for which it was acquired is inoperative;

   (3) Reproduced for safekeeping (archives) or backup purposes;

   (4) Modified, adapted, or combined with other computer software, provided that the modified, combined, or adapted portions of the derivative software incorporating restricted computer software are made subject to the same restricted rights;

   (5) Disclosed to and reproduced for use by support service contractors in accordance with subparagraphs (b) (1) through (4) of this clause, provided the Government makes such disclosure or reproduction subject to these restricted rights; and

   (6) Used or copied for use in or transferred to a replacement computer.

   (c) Notwithstanding the foregoing, if this computer software is published copyrighted computer software, it is licensed to the Government, without disclosure prohibitions, with the minimum rights set forth in paragraph (b) of this clause.

   (d) Any other rights or limitations regarding the use, duplication, or disclosure of this computer software are to be expressly stated in, or incorporated in, the contract.

   (e) This Notice shall be marked on any reproduction of this computer software, in whole or in part.

Contents

Preface

Who Should Read This Guide xi
Prerequisites for Developers xii
How to Use This Guide xii
Typographical Conventions xiv
Related Documentation xiv

Chapter 1 Introduction

SEI Toolkit Versioning 1-1
SEI Toolkit Overview 1-2
Installing the Toolkit 1-3
SEI Toolkit Goals 1-3

Chapter 2 Toolkit Architecture

Developing SPECTRUM Extensions 2-1
- Development Phase 2-2
- Manufacturing Phase 2-3
- Integration Phase 2-3

Development Phase 2-3
Manufacturing Phase 2-4
Integration Phase 2-5

Chapter 3 Index Files

Introduction 3-1
Index File Syntax Rules 3-1
- General Index File Syntax 3-2
- Environment Variables 3-2
- Escaping Special Characters 3-3

Index File Entries 3-5
Index File Platforms 3-5
Index File Entry Definitions 3-6
- Extension Module Description Entries 3-6
- Extension Module Relation Entries 3-10
- File Distribution Labels 3-15
- Cabletron-Reserved Labels 3-23

Index File Entry Examples 3-23

Extension Module Description Label Examples 3-25
Example Line 1 – mm: 3-25
Example Line 2 – vend: 3-25
Example Line 3 – rev: 3-26
Example Line 4 – irev: 3-26
Example Line 5 – level: 3-26
Example Line 6 – plat: 3-26
File Distribution Entry Examples 3-26
Example Line 7 – dep: 3-27
Example Line 8 – comp: 3-27
Example Line 9 – file: Install mmdesc 3-27
Example Line 10 – file: SG iib 3-28
Example Line 11– file: SS db 3-29
Example Line 12 – file: SS ih 3-30
Example Line 13 – file: SS cus 3-31
Example Line 14 – file: SS doc 3-32
Example Lines 15–20 – deflt: 3-33
Example Lines 21–27 – deflt: 3-34
Example Line 28– file: SS db 3-36
Index File Conversion 3-37
Introduction 3-37
Manual Index File Conversion 3-38
Automatic Index File Conversion 3-39
Index File Conversion Example 3-39

Chapter 4 Tools

Special Requirements 4-1
SEI Toolkit Version Compatibility 4-1
SEI Toolkit and Windows NT 4-1
SEI Toolkit Tools 4-2
Development Tools Specification 4-2
Development: mkmmconv 4-2
Development: mkmmdeps 4-5
Development: mkmm 4-7
Retired Features: 4-10
mkmm.sys and mkmm.loc 4-11
Source File Extension Handling in mkmm and mkmmdeps 4-13
Manufacturing Tools Specification 4-15
Manufacturing: mkmanf 4-15
Updating the MMML File 4-18
Example for Updating the MMML File 4-19
Retired Features: 4-20
Manufacturing: mkarea 4-20
Manufacturing: mkdist 4-23
Retired Features: 4-27
SPECTRUM Distribution Media 4-29
Virtual Tape (Vtape) 4-29
Vtape Creation Example 4-29

CD-ROM Virtual Tape (CD-ROM Vtape) 4-30
CD-ROM Vtape Creation Example 4-31
Integration Tools Specification 4-32
Integration: Install 4-32
Integration: Install.quick 4-32

Chapter 5 Shippable Files

Introduction 5-1
Levels, Products and File Types 5-1
Development of Shippable Files 5-3
Distribution of Shippable Files 5-5
Installation of Shippable Files 5-8
Special Shippable Files 5-11
MM Description Files 5-11
Introduction 5-11
Writing MM Description Files 5-11
Shipping MM Description Files 5-12
Custom Install Scripts 5-13
Introduction 5-13
Uses of Custom Install Scripts 5-14
Writing Custom Install Scripts 5-16
Shipping Custom Install Scripts 5-19
Custom Install Script Predefined Environment Variables 5-20
Predefined Functions for Custom Install Script 5-23
Retired Custom Install Script Variables and Functions 5-24
X Resource File Templates 5-26
X Resource File Template Creation 5-26
SPECTRUM Graphical Application Standards 5-29
stdmenu and spmamap files 5-30
PIB Contribution Files 5-30
Manual Generation of PIB Contribution Files 5-31

Chapter 6 Use Scenarios

Scenario 1: MM Vtape Distribution 6-1
Installing the SEI Toolkit 6-2
Setting Up a Development Area 6-2
Creating an Index File 6-3
Creating an MM Archive File 6-6
Creating the Manufacturing Area 6-7
Adding the MM Archive File to the Manufacturing Area 6-7
Distributing the management module to Vtape 6-8
Scenario 2: Developer's Tools Integration 6-9

Appendix A Errors and Warnings

Output Messages A-1
mkmmconv Errors A-2
mkmm and *mkmmdeps* Errors A-6

Glossary

Glossary of Terms for SEI Toolkit Glossary-1
General Glossary of Terms for SPECTRUM Glossary-6
Common SPECTRUM Acronyms Glossary-12

Index

Figures

Chapter 2 Toolkit Architecture

Figure 2-1. SPECTRUM Extensions Integration Process 2-2

Chapter 3 Index Files

Figure 3-1. Entries in the SMACME-MM.i Index File (Fictitious) 3-24
Figure 3-2. Text for DENET.i Index File (Fictitious) 3-40
Figure 3-3. Text for DC NET.i.conv Index File (Fictitious) 3-41
Figure 3-4. Contents of DCNET.p File (Fictitious), as Generated by mkmmconv 3-41

Chapter 5 Shippable Files

Figure 5-1. FOO.cus Custom Install Script 5-17
Figure 5-2. FOO.i Custom Install Script Index File (Fictitious) 5-19
Figure 5-3. Script Example Showing Use of $AUTO Environment Variable 5-21
Figure 5-4. Script Example Showing Use of $CPUTYPE Environment Variable 5-21
Figure 5-5. Sample Custom Script for Installation of xappl Application 5-28

Chapter 6 Use Scenarios

Figure 6-1. Contents of /usr/manmods/devarea Development Area for Scenario 1 6-3
Figure 6-2. Index File SM-MYR1001.i for Extension Module SM-MYR1001 6-6
Figure 6-3. Representative init_env file for Scenario 2 (Using Bourne shell) 6-10

Tables

Chapter 3 Index Files

Table 3-1. Standard Platforms for SPECTRUM Extension Modules 3-6
Table 3-2. Significance of Basic <snamepat> Patterns 3-21
Table 3-3. Old-Style Index File Entries and New-Style Equivalents 3-38

Chapter 5 Shippable Files

Table 5-1. Relation of File to Installed SPECTRUM Product by Index File Level and Product 5-2
Table 5-2. Preferred Development Methods for Shippable Files 5-3
Table 5-3. File Statistics by Product and Type 5-6
Table 5-4. Special Processing of Shippable Files by Type 5-8

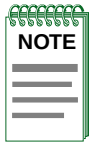
Preface

This document provides information for using the SPECTRUM Extensions Integration Toolkit in conjunction with SPECTRUM Level I and Level II Developer's Tools, identifying relationships pertaining to changes effective with SPECTRUM Release 5.0.

Cabletron's SPECTRUM network management system originally was designed for a known collection of hardware components and management functions, as stored in its knowledge base. That original program can be "extended" through the use of management modules (MM files), which give that knowledge base facts and relationships pertaining to new components and/or new management functions pertaining to associated managed nodes. These management modules are also called extension modules, or simply extensions. The SPECTRUM Extensions Integration Toolkit, hereinafter called simply the SEI Toolkit, is one of the SPECTRUM Level I Developer's Tools, designed to enable customer and VAR developers to extend the functionality of SPECTRUM by creating and installing custom modules.

Who Should Read This Guide

This guide is intended for developers who want to create integrated and installable SPECTRUM extension modules for customer use or resale.



The internal formats of the MM archive, manufacturing area, and SPECTRUM media provided in this document are published for informational purposes only. These formats are subject to change without notice, and Cabletron reserves the right to discontinue support of any such published internal format at any time. Cabletron neither supports nor recommends creation or modification of MM archives, manufacturing areas, or SPECTRUM media by the use of utilities or methods other than those provided in the SEI Toolkit.

Prerequisites for Developers

Prior to attempting to build SPECTRUM extension modules using the SEI Toolkit, developers should have significant exposure to SPECTRUM. They should also read the ***SPECTRUM Concepts Guide*** to familiarize themselves with the underlying foundation of SPECTRUM extension modules before attempting to create extension modules. Developers are also expected to have an understanding of the important concepts of the object-oriented paradigm, such as data encapsulation, inheritance, and polymorphism. SPECTRUM install programs make extensive use of UNIX® shell scripts; an understanding of scripts and experience writing scripts can be very beneficial. Experience with UNIX, the X Window System™, and the OSF/Motif™ user interface are additional pluses, since SPECTRUM is based on these systems.

How to Use This Guide

This guide describes how to use the SEI Toolkit to package, integrate, and install new SPECTRUM customer-written or VAR-written extension modules and client applications. It is organized as follows:

Chapter	Description
Chapter 1 *Introduction*	This chapter explains the Cabletron version number convention, identifies the need to convert pre-4.0 format files to SPECTRUM 5.0 requirements, and provides a brief overview of the SPECTRUM Extensions Integration Developer's Toolkit and the three categories of SPECTRUM products (end-user, Level I developer tools, and Level II developer tools).
Chapter 2 *Toolkit Architecture*	This chapter first gives an overview of the three categories of software tools (Development, Manufacturing, and Integration) that constitute the elements of the SPECTRUM Extensions Integration Toolkit. The subsequent discussion outlines the different phases of the overall process of developing an extension module.

Chapter	Description
Chapter 3 *Index Files*	This chapter describes SPECTRUM Extensions Integration Toolkit index file formats, conversions, and usage. The discussion includes detailed definitions of all recognized index file entries (covering description entries and relation entries), and identifies Cabletron-reserved entries. This chapter also identifies and describes all file distribution labels, gives a representative example of a file entry with an accompanying line-by-line detailed analysis, and identifies and explains pib: error messages. Finally, this chapter also provides rules and hints for converting old-style index files into new-style index files, complete with a before-and-after example showing what gets changed.
Chapter 4 *Tools*	This chapter describes the tools provided by the SEI Toolkit and explains their requirements, options, and usage, as well as indicating "retired" features. The discussion also includes examples for creating various types of SPECTRUM media.
Chapter 5 *Shippable Files*	This document provides information for using the SPECTRUM Extensions Integration Toolkit in conjunction with SPECTRUM Level I and Level II Developer's Tools, identifying relationships pertaining to changes effective with SPECTRUM Release 5.0.
Chapter 6 *Use Scenarios*	This chapter illustrates two typical scenarios of SEI Toolkit use, showing its flexibility in the type of components that you can install, configure, and operate with SPECTRUM. The first scenario demonstrates the process of accomplishing MM vtape distribution. The second scenario illustrates how you can integrate other SPECTRUM Level 2 Tools.
Appendix A *Errors and Warnings*	This appendix contains descriptions of error messages (including warnings and informational messages) generated by mkmmconv, mkmmdeps, and mkmm tools in the SEI Toolkit.
Glossary	This glossary defines terms related to the manufacturing/distribution/integration processes for SPECTRUM extension modules.

Typographical Conventions

Certain typographical conventions are used throughout this document to distinguish program names, variables, etc., as follows:

- Code examples and screen messages appear in `Regular Courier type` if showing system display or output, in **`Bold Courier`** if showing what you should type into the system. Where separation from the surrounding text seems called for, these computer-text passages may be shown within a shaded frame (colored light green in the online version).
- Cross-references to other sections, tables, or figures in this publication usually include the actual page number and when appropriate give the applicable *titles in italic* text. If you are viewing an online version of the document, the applicable identification portion of these cross-references will be colored blue; these blue entries are hypertext hotspots, and clicking on them will take you directly to the referenced location.
- Titles of other Cabletron publications appear in ***bold italic*** text.
- Executable script and program names, along with variables, appear in `Regular Courier`. For example, a program named “Application” would appear as `Application`.
- Keyboard labels and display-screen nomenclature are given in boldface type—such as the **Return** key and the **OK** button.
- Emphasis is designated by *italics*.

Related Documentation

The following Cabletron documents contain further information on topics discussed in this guide:

- ***Bridging Applications***
- ***Defining Resources***
- ***Getting Started with SPECTRUM for Operators***
- ***How to Manage Your Network with SPECTRUM***
- ***MIB II Applications***
- ***MIBs and the Application View***
- ***Miscellaneous Applications***
- ***SpectroSERVER Application Programming Interface Developer’s Guide***
- ***SpectroSERVER Application Programming Interface Reference***

- ***SPECTRUM Annotation Toolbox***
- ***SPECTRUM API Developer's Guide***
- ***SPECTRUM Concepts Guide***
- ***SPECTRUM Database Management***
- ***SPECTRUM GIB Editor Guide***
- ***SPECTRUM Global Classes Reference***
- ***SPECTRUM Icons***
- ***SPECTRUM IIB Editor Guide***
- ***SPECTRUM Inference Handler API Developer's Guide***
- ***SPECTRUM Knowledge-Base Guide***
- ***SPECTRUM Menus***
- ***SPECTRUM Model Type Editor User's Guide***
- ***SPECTRUM MSAP/EPI Developer's Guide***
- ***SPECTRUM Partner's Guide to Integrating Applications***
- ***SPECTRUM Report Generator User's Guide***
- ***SPECTRUM User Security and User Maintenance***
- ***SPECTRUM Views***
- ***SPECTRUM VnmParmBlock Reference***

Questions About SPECTRUM Documentation?

Send your questions, comments or suggestions regarding SPECTRUM documentation to the Technical Communications Department directly via the following internet address:

spectrum-techdocs@ctron.com

Chapter 1

Introduction

This chapter explains the Cabletron version number convention, identifies the need to convert pre-4.0 format files to SPECTRUM 5.0 requirements, and provides a brief overview of the SPECTRUM Extensions Integration Developer's Toolkit and the three categories of SPECTRUM products (end-user, Level I developer tools, and Level II developer tools).

SEI Toolkit Versioning

Cabletron reserves the right to modify external interfaces and internal behaviors of tools in the SEI Toolkit in any way during any official release, major or minor, of SPECTRUM. In particular, Cabletron does not guarantee backward compatibility of tools in the SEI Toolkit across any releases, major or minor, of SPECTRUM.

The version number of an official release of the SPECTRUM enterprise management product is normally given as two numbers separated by a decimal point (for example, 5.0). The number before the decimal point is the major release number, and the number after the decimal point is the minor release (or point release) number. Thus, SPECTRUM 5.0 is major release 5, minor release 0, of SPECTRUM.

An increment of the major release number represents a major release of SPECTRUM with the minor release number at that time being set to 0. SPECTRUM 4.0, then, was a major release over the previous SPECTRUM 3.1 version, and SPECTRUM 5.0 has occurred without any intervening minor release. In the event of a minor release, the minor release number is incremented (for example, SPECTRUM 5.1 would be a minor release over the SPECTRUM 5.0 release).

The version number of the SPECTRUM Extensions Integration Developer's Toolkit (SEI Toolkit) is the same as the version number of SPECTRUM shipped on the same SPECTRUM medium. Thus, version 5.0 of the SEI Toolkit is shipped with SPECTRUM 5.0.

In general, Cabletron endeavors to maintain backward compatibility of the SEI Toolkit across minor releases, but not across major releases. Thus, unless otherwise stated, you may assume that files and structures (for example, extension module archives and manufacturing areas) generated by any given release of the SEI Toolkit will be compatible with subsequent minor releases of the SEI Toolkit, but there is no guarantee that this condition will remain valid for subsequent major releases.

In particular, extension module archives and manufacturing areas generated using version 3.1 and earlier of the SEI Toolkit are not compatible with version 4.0 or subsequent versions of the SEI Toolkit, including this SPECTRUM 5.0 release.

As a consequence, in order to achieve compatibility with SPECTRUM 4.0, users needed to convert any index files that were created for SPECTRUM 3.1 or earlier formats and then rebuild any extension module archives and manufacturing areas. That requirement continues for SPECTRUM 5.0.

Failure to convert existing index files to the SPECTRUM 4.x format will result in inability to properly integrate your extension modules into SPECTRUM 5.0 areas.

SEI Toolkit Overview

The SPECTRUM Extensions Integration Developer's Toolkit (SEI Toolkit), Cabletron Part Number ST-CSI1003, provides high-level tools to assist SPECTRUM customers and VARs (Value Added Resellers) in packaging and installing SPECTRUM extension modules.

SPECTRUM extension modules are software modules that extend the functionality of the SPECTRUM line of products. There are three basic types of SPECTRUM extension modules:

- Core components, which support standard SPECTRUM network management functionality.
- Management modules, which may be integrated into SPECTRUM to support management of new network devices.
- External applications, which are independent tools that work in conjunction with SPECTRUM to enhance its usability.

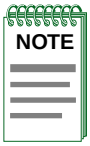
Core components are developed exclusively at Cabletron, while many customers and VARs are involved in the development of management modules and external applications to extend the capabilities of SPECTRUM to meet their specific networking needs. Cabletron provides an array of toolkits, which greatly facilitate the development of SPECTRUM extension modules.

Once a SPECTRUM extension module has been developed, it must be put on a storage medium, transported to a customer site, and installed. The installer should be confident that the installation will properly integrate the extension module with those SPECTRUM products already installed at the site. This is the purpose of the SEI Toolkit.

The SEI Toolkit contains a collection of tools that support the proper and complete distribution and installation of customer and VAR-developed SPECTRUM extension modules. The SEI Toolkit is required for the development of all SPECTRUM extension modules that will be Cabletron-certified and sold to customers.

Installing the Toolkit

To install the SEI Toolkit, use `Install`, the SPECTRUM installation utility. `Install` is used to install all SPECTRUM core components, extension modules, and external applications. The `Install` program first retrieves the SEI Toolkit files from the distribution medium (CD-ROM or virtual tape) and then calls the custom script, `insdk.cus`, to build the correct directory structure and install the toolkit files. For more information on `Install` and installing SPECTRUM components, refer to the ***SPECTRUM Installation Guide***.



Before installing the SEI Toolkit, you must install SPECTRUM in accordance with the instructions in the ***SPECTRUM Installation Guide****.*

SEI Toolkit Goals

The primary goal of the SEI Toolkit is to provide customers and VAR developers with tools to integrate SPECTRUM-compatible extension modules with SPECTRUM. As with other interfaces, product packaging and integration require standardization to ensure interoperability. The principles and tools described in this document are designed to ease the integration of components from different vendors into a SPECTRUM system. By providing a standard means of installing and integrating components into SPECTRUM and making these integration tools available to customers and VARs, the SEI Toolkit:

- **Reduces time to market for SPECTRUM VAR developers** – Less time is needed to develop installation software and test its compatibility with software from Cabletron and other vendors.

- **Facilitates integration of VAR-produced components** – Since all Cabletron development partners use the same tools to produce integrated extension modules to SPECTRUM, these products are less likely to cause installation or integration compatibility problems for customers when used individually or together.
- **Reduces the investment required for developing SPECTRUM extension modules** – This follows not only from the reduction in time-to-market and a decrease in compatibility issues, but also because the software and expertise used to produce an extension module can be applied to multiple products, across multiple platforms and multiple releases.
- **Increases the value of SPECTRUM extension modules to customers** –Customers benefit from interoperable products from different vendors. Whether they buy extension modules off the shelf or develop their own, customers know that these extension modules will work together.

In general, all the SPECTRUM Developer's Tools are designed to place minimal constraints on a VAR's or customer's development environment. The SPECTRUM Developer's Tools are intended to integrate into existing environments, thereby providing maximum advantage in the development of SPECTRUM-compatible products.

Chapter 2

Toolkit Architecture

This chapter first gives an overview of the three categories of software tools (Development, Manufacturing, and Integration) that constitute the elements of the SPECTRUM Extensions Integration Toolkit. The subsequent discussion outlines the different phases of the overall process of developing an extension module.

Figure 2-1 shows an overview of the SPECTRUM Extensions Integration Process from development to customer site integration.

The SEI Toolkit provides several utilities for developers of SPECTRUM-compatible extension modules. These tools are organized into three groups: Development, Manufacturing, and Integration. Each group is associated with a particular phase of a product's evolution, as follows:

Development	Developing and testing compatible extension modules.
Manufacturing	Packaging and preparing extension modules for distribution.
Integration	Installing and merging extension modules into SPECTRUM.

Developing SPECTRUM Extensions

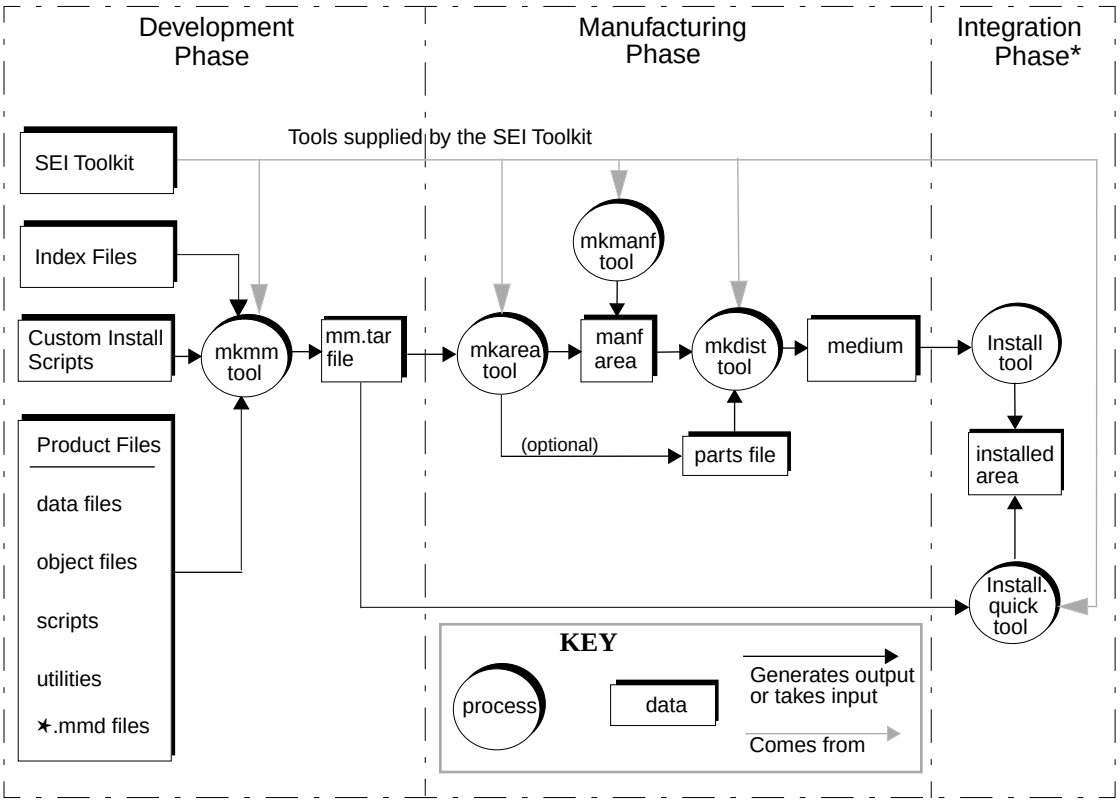
To help you understand how the SEI Toolkit is used during the development of SPECTRUM extension modules, the following paragraphs outline the phases you would perform in the product evolution process. This discussion identifies the use of the various SEI tools at different steps within each phase.

Development Phase

The development phase involves three sequential steps:

1. Design and develop the files that are to be shipped and installed as part of the extension module. (Cabletron provides a host of development tools for use in developing files for Level 1 extension modules. Many of these tools are described in *Development of Shippable Files*, starting on page 5-3.)

Figure 2-1. SPECTRUM Extensions Integration Process



* The integration phase involves installation and integration of extension modules, using `Install`. For instructions, refer to the **SPECTRUM Installation Guide**.

2. Create an *index file* that includes descriptive information about the extension module, along with a list of files to be shipped and installed as part of the extension module.

3. Use the ***mkmm*** tool and the index file to create an *extension module archive file* for the extension module.

Manufacturing Phase

The manufacturing phase involves three sequential steps:

1. Use the `mkmanf` tool to create a *manufacturing area*.
2. Copy the extension module archive file (mm.tar) to the manufacturing area, then use the `mkarea` tool to add the archive file into the manufacturing area.
3. Use the `mkdist` tool to create a *SPECTRUM* virtual tape (tape image on disk), containing selected extension modules from the manufacturing area.

Integration Phase

Use the `Install` tool to install selected extension modules from the SPECTRUM distribution medium into an existing *SPECTRUM installation area* at a customer site, integrating those modules with the existing files. You may also use the `Install.quick` tool to install extension module archive files directly into a SPECTRUM installation area at the development site for testing purposes.

Development Phase

The development environment is the context in which a VSAR (Value Added Reseller) produces SPECTRUM-compatible extension modules. A development environment at a VAR site usually includes:

- Directory Structure
- Environment Variables
- Text Editors and Word Processors
- Compilers
- Source Control Systems
- Test Tools
- CASE Tools or Analysis Tools

A development area (a directory structure containing files) depends on the development environment being used. For a given SPECTRUM toolkit, the development area provides the directory structure and tools needed for the development of extension components. The development phase described here assumes that a separate development area is defined for the SPECTRUM Developer's Tools products in use at the VAR site. The actual structure of the development area for a given SPECTRUM toolkit is not dictated by the SPECTRUM Developer's Tools; it is left to the judgment of the developers at the VAR site.

The purpose of the SEI Toolkit is to provide tools to package components produced by the other SPECTRUM toolkits. These tools assume that the components required by the SEI Toolkit have been produced by prior use of the SPECTRUM developer's other tools.

Beyond these guidelines for setting up development areas, and presuming the use of the SPECTRUM Developer's Tools, the SEI Toolkit allows packaging of SPECTRUM extension modules in a way that is as independent of the development environments (and component locations) as possible (thus minimizing the impact on the development environment).

Instead, SPECTRUM Developer's Tools complement the development environment already in place at the VAR site by providing interfaces into SPECTRUM extension capabilities.

During the development phase, the user must supply several components in the development area in preparation for the Manufacturing phase. These components include:

- *Index files* (text files that detail the exact contents of a SPECTRUM Extension module, in a format defined by Cabletron).
- *MM archive files* (`mm.tar` - Management Module archive files).

Index files are produced according to the format described in Chapter 3, *Index Files*. The index file and the SEI development phase tools then produce Management Module archive files from component files; this process is described in Chapter 4, *Tools*, which provides practical details for distributing and integrating extension modules.

When a SPECTRUM extension module has completed development testing and is ready for transfer to distributable media, use the "check" option of `mkmm` to stamp and checksum the `CsGib` and `CsIib` files that are to be distributed with the extension module. The "check" option is a way for subsequent component integrations to determine if some standard components have been modified to provide local customization. Cabletron recommends the use of this option prior to delivery of a VAR-developed extension module to customers, to ensure that later versions of the installation utility preserve a customer's work. Once `mkmm` is finished, the extension module archive (`mm.tar`) files can enter the manufacturing phase. This event completes the development process for the extension module, with the exception of additional testing.

Manufacturing Phase

The manufacturing phase occurs when SPECTRUM extension components are ready for release or release testing. Manufacturing is identified as a separate part of the product development evolution because it is often performed by other parts of a software development organization.

This phase is referred to as “manufacturing” because the actual packaging of the components on the distribution media occurs at this time. The tools provided for manufacturing allow a VAR to develop and release SPECTRUM extension modules separately or together. The manufacturing process provided by the SEI Toolkit is flexible enough that each different customer order can be filled on a customized basis. For example, Customer 1 may purchase extension modules A and B, while Customer 2 purchases extension modules B and C. The Manufacturing tools allow the VAR to produce a custom distribution for each customer. For high volume products, it may be desirable to duplicate the distribution media rather than produce them individually. The SEI Toolkit does not provide high-speed duplication tools, however; they must be provided by the VAR.

Integration Phase

In the Integration Phase, you install the SPECTRUM extension module at a customer site and integrate it with an existing SPECTRUM system. To do this, you use the SPECTRUM installation utility, `Install` (or `Install.quick`, which bypasses the manufacturing phase). This utility installs all extension modules included in the distribution and automatically executes any associated custom installation scripts. The SEI Toolkit utilities guarantee that the proper version of `Install` is included on every SPECTRUM medium. The installation process is described in detail in the ***SPECTRUM Installation Guide***, provided with the initial purchase of SPECTRUM.

Chapter 3

Index Files

This chapter describes SPECTRUM Extensions Integration Toolkit index file formats, conversions, and usage. The discussion includes detailed definitions of all recognized index file entries (covering description entries and relation entries), and identifies Cabletron-reserved entries. This chapter also identifies and describes all file distribution labels, gives a representative example of a file entry with an accompanying line-by-line detailed analysis, and identifies and explains pib: error messages. Finally, this chapter also provides rules and hints for converting old-style index files into new-style index files, complete with a before-and-after example showing what gets changed.

Introduction

An index file is an editable text file that defines an extension module. The index file contains descriptive information about the extension module itself, as well as a list of the files that are shipped and installed as part of the extension module. The `mkmm` tool uses the index file to locate, reorganize, and package the shippable files into a single unit, called an extension module archive (currently produced in UNIX `tar` archive format). The extension module archive can then be added to a manufacturing area, placed on a distribution medium, shipped to a customer site, and installed.

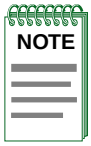
Index File Syntax Rules

The following discussion outlines the general rules for index files and then provides information for the special rules pertaining to environment variables and special characters.

General Index File Syntax

The following general syntax rules apply to SPECTRUM 4.x/5.0 index files:

- Each index file entry consists of zero or more fields of visible characters, which may be group-separated by one or more whitespaces (space, tab, carriage return, or formfeed). Whitespaces may also appear at the beginning or end of an index file entry.
- Index file entries are newline-delimited (that is, one entry per line, terminated either by a newline character or by the end of the file). There is no mechanism for continuing an index file entry across more than one line. There is no practical limitation on the length of index file entries.



Because of the <115-character count limitation of the width of the page image area in this publication, foldover lines in reproduced or simulated program coding used as examples in this document include the backslash character to show line folding in accordance with standard UNIX convention in order to show the entire contents of the simulated text. In an actual index file, however, you cannot use a backslash to continue an entry across more than one line.

- The following characters are legal in an index file:
- Visible characterASCII 33 (!) through 126 (~)
  SpaceASCII 32
  TabASCII 9
  Newline (Linefeed)ASCII 10
  Carriage ReturnASCII 13 (^M)
  FormfeedASCII 12 (^L)
- Each line of an index file consists of an index file entry, which may conclude with an optional comment; if included, the comment begins at the first pound sign (#) in the line and extends to the next newline or end-of-file character.
- If an index file entry has no fields, the entire line is considered a comment line. Otherwise, the first field of an index file entry must be a valid index file label, as identified in *Index File Entries*, starting on page 3-5. Any other index file entry is illegal.

Environment Variables

Environment variables of the form $VAR, ${VAR}, or $(VAR) may appear in index files, where VAR is a standard shell variable name (one or more alphanumerics and underscores, not beginning with a digit). When `mkmmdeps` or `mkmm` processes an index file, that processing expands embedded environment variables to their values, as exported by the invoking shell. It is

therefore necessary to export any environment variables used in an index file prior to invoking `mkmmdeps` or `mkmm` to process that index file.

- For example, if the environment variable `$VNUM` is set to the value "`1.1`" and exported before either `mkmmdeps` or `mkmm` is invoked, the index file line:

```
mm: Acme Router Version $VNUM
```

is interpreted as the index file entry:

```
mm: Acme Router Version 1.1
```

Environment variables having special meaning to any particular shell (`$_`, `$PATH`, `$PWD`, etc.) are not guaranteed to expand properly, and their use in index files should be avoided. Environment variables not of the form `$VAR`, `${VAR}`, or `$(VAR)`, (`$$`, `$!`, `$?`, `$0`, `$1`, etc.) are not expanded within index files.

Escaping Special Characters

The pound sign (**#**) and the dollar sign (**$**) have special meaning within an index file. These characters must be "escaped" to take on their literal value within an index file entry.

- The occurrence of a pound sign in an index file always denotes the beginning of a comment. The pound sign and any subsequent characters on the line are ignored when either `mkmmdeps` or `mkmm` processes the index file. To prevent interpretation of the pound sign as the beginning of a comment, replace the pound sign with the environment variable `${PS}`, which `mkmm` will expand to a literal pound sign.

  For example, the index file line:

```
mm: Acme Router #2 Management Module
```

  is interpreted by `mkmm` or `mkmmdeps` as the index file entry "**`mm: Acme Router`**" followed by the comment "**`#2 Management Module`**," whereas the line:

```
mm: Acme Router ${PS}2 Management Module
```

  will be interpreted as the index file entry:

```
mm: Acme Router #2 Management Module
```

- The dollar sign, in the context $VAR, ${VAR}, or $(VAR), denotes the beginning of an environment variable. When either mkmmdeps or mkmm processes the index file, environment variables are expanded and the delimiting dollar sign is lost. To prevent interpretation of a dollar sign as part of an environment variable, replace the dollar sign with the environment variable ${DS}, which either mkmmdeps or mkmm expand to a literal dollar sign.

  If the environment variable $VNUM is set to the value "1.1" and exported before mkmmdeps or mkmm is invoked, for example, the index file line:

```
mm: Acme Router Version $VNUM
```

  is interpreted as the index file entry:

```
mm: Acme Router Version 1.1
```

  whereas the entry

```
mm: Acme Router Version ${DS}VNUM
```

  is interpreted as the index file entry:

```
mm: Acme Router Version $VNUM
```

  The above example also illustrates the use of the ${VAR} or $(VAR) forms of an environment variable to separate environment variable names from the immediately following text. For instance, in the index file line:

```
mm: Acme Router Version $VNUM
```

  the environment variable $VNUM is expanded. Conversely, in the line

```
mm: Acme Router Version ${V}NUM
```

  the environment variable ${V} is expanded, while NUM keeps its literal value.

Index File Entries

An index file is made up of entries and comments — one each per line. Each entry is made up of fields separated by white spaces.

Three types of index file entries may appear in an index file: “old-style,” “new-style,” and “Cabletron-reserved.” Old-style index file entries are those that were supported prior to this version of the SEI Toolkit but are not expected to be supported beyond this version. New-style entries are those that are supported in this version of the SEI Toolkit and are expected to be supported in subsequent versions, as well. Cabletron-reserved index file entries are reserved for use in Cabletron-developed extension modules only.

Almost all of the old-style entries used for shipping files in extension modules can be replaced with new-style entries. In addition, new-style entries provide new functionality for shipping files within extension modules, including:

- Independent specification of source and installed locations of shippable files.
- Shipment of files only for specified platforms.
- Specification of default locations for batches of source or installed files.

New-style entries are described in *Extension Module Description Entries*, starting on page 3-6, and in *Extension Module Relation Entries*, starting on page 3-10. Old-style labels, and their new-style equivalents, are listed in *Manual Index File Conversion*, starting on page 3-38. Cabletron-reserved index file labels are listed in *Cabletron-Reserved Labels*, starting on page 3-23.

The `mkmm` tool supports conversion of index files containing old-style entries to index files containing only new-style entries. This process is described in *Index File Conversion*, starting on page 3-37.

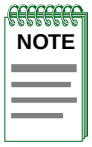
Index File Platforms

Distribution and installation of SPECTRUM 5.0 extension modules are supported on several platforms (hardware and operating system configurations). Whenever referring to these platforms in index files, you should use the applicable standard platform names, as delineated in Table 3-1.

Table 3-1. Standard Platforms for SPECTRUM Extension Modules

Vendor	OS	Name
Sun Microsystems	Solaris 2.5.1	sun4c
Microsoft	NT 4.0	intel

Note that these names are specific to the SEI Toolkit and do not refer to any specific OS or hardware type of any platform. In particular, sun4c refers to any Solaris platform on which SPECTRUM 4.x and 5.0 distribution and installation are supported, including machines with hardware types sun4c, sun4m, and others.



SPECTRUM 5.0 does not support Sun Microsystems' SunOS (previously referenced as type sun4). Do not attempt to create or distribute management modules or other extensions pertaining to sun4, as no installation software is available.

Index File Entry Definitions

Each entry in an index file must contain an index file label followed by an appropriate number of data fields. The label generally indicates the meaning and purpose of the entry, and each entry must conform to the syntax associated with its label.

The following subsections provide full descriptions for all index file entries, grouped into three label categories: extension module description labels, file distribution labels, and Cabletron-reserved labels. The index file entries within each category are arranged alphabetically, with the applicable label being identified in the shaded block at the beginning of the discussion. For each label, this discussion gives the entry syntax, a brief example, the entry description, and applicable conventions. For "how to" examples of index file entries, or for an explanation of index file entry syntax restrictions that apply to all labels, refer to *Index File Entry Examples*, starting on page 3-23.

Extension Module Description Entries

Extension module description entries describe the extension module associated with the index file. Each extension module description entry should appear *exactly once* in each index file. If an extension module index file does not contain an extension module entry or contains more than one such entry,

`mkmm` will produce an error warning when creating the extension module archive, and `Install` may install the extension module improperly.

irev:

Syntax: `irev: <version>`

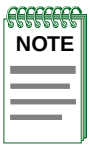
Example: `irev: 04.00.00.000`

Description: The `irev:` entry provides the machine-friendly version number <version> of the extension module. If two versions of an extension module have different `irev:` version numbers, the one with the smaller number is the older of the two.

The <`version`> identification must use the format "dd.dd.dd.ddd," where each "d" is a decimal digit.

The main use of the `irev:` label is to identify any instance of extension module version downgrading during installation. Extension module version downgrading occurs when an older version of an extension module is installed into an area that already contains a newer version of that same extension module. Normally, version downgrading is undesirable. Sometimes you may want to downgrade an extension module, however—for instance, when the newer installed version has a bug that was not present in the older version.

`Install` detects a downgraded extension module by comparing the `irev:` version numbers of the previously installed extension module with the one being installed. If the previously installed extension module has a larger `irev:` version number, an older version of the extension module is being installed over a newer version and a downgrade is reported for that extension module.



If you omit the `irev:` *label entry,* `Install` *will identify the extension module as a very old version that predated the use of the* `irev:` *label entry.*

level:

Syntax: `level: <level>`

Example: `level: 1`

Description: The `level:` entry specifies the level of the extension module. The extension module level (also known as the MM level) indicates how software in the extension module is related to SPECTRUM (SpectroSERVER and/or SpectroGRAPH). The <level> designation must be one of the following three number values:

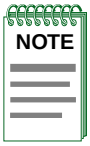
0 The software in the extension module is a *required* part of SPECTRUM (either for SpectroSERVER and/or SpectroGRAPH). A Level 0 extension module is often referred to as a SPECTRUM *core module*. The right to develop and ship Level 0 extension modules is reserved to Cabletron, alone.

1 The software in the extension module is an *optional* part of SPECTRUM. A Level 1 extension module is often referred to as a *management module*, typically containing software that helps SPECTRUM manage a particular device or class of devices. While Cabletron develops and ships a wide variety of Level 1 extension modules, many others are developed by SPECTRUM users for their own management needs, or by VARs for resale to other SPECTRUM users.

2 The software in the extension module is *not* part of SPECTRUM (although it may work in conjunction with SPECTRUM). A level 2 extension module is often referred to as an *external application* or *toolkit*, and it typically contains a standalone tool or demo for use in the SPECTRUM environment. Most level 2 extension modules are developed at Cabletron.

`mkdist` uses the extension module level value to select extension modules to put on the distribution medium.

`Install` uses the extension module level value to select extension modules for installation. In addition, the extension module level also affects the way in which files in an extension module are processed during installation. Since Level 0 and Level 1 extension modules are considered part of SPECTRUM (SpectroSERVER or SpectroGRAPH), files shipped in Level 0 and 1 extension modules undergo special processing during installation to integrate them into SPECTRUM. Whereas level 2 extension modules are not considered part of SPECTRUM, files contained in level 2 extension modules are not integrated into SPECTRUM during installation. For more detailed information on how the extension module level affects installation of specific files, see *Installation of Shippable Files*, starting on page 5-8.



Do not confuse extension module levels (also known as MM levels) with toolkit levels.

There are two types of toolkits, respectively called Level 1 and Level 2 toolkits, which are used to customize or otherwise modify the SPECTRUM environment. The distinction is that Level 1 toolkits can be used by any SPECTRUM user, whereas Level 2 toolkits require sophisticated programming knowledge, including familiarity with C++.

All toolkits are standalone applications, which are not integrated into SPECTRUM. Consequently, all toolkits, whether Level 1 or Level 2, are shipped as level 2 extension modules, and the associated extension module index file should contain the entry "`level: 2`*").*

mm:

Syntax: `mm: <name>`

Example: `mm: Extensions Integration Toolkit`

Description: The `mm:` entry specifies a descriptive name for the extension module.

- `<name>` may consist of a single word or several words, but should be kept to a total of no more than 30 characters, including spaces, in order to be visible on the screen when displayed as part of a message.
- `<name>` should be descriptive — that is, "Model Type Editor" is preferable to "ST-MTE1000."

`Install` displays this `<name>` identification on the MM selection screen to aid the operator in selecting extension modules.

rev:

Syntax: `rev: <version>`

Example: `rev: 3.0rev1`

Description: The `rev:` entry provides the human-friendly mnemonic version number of the extension module.

- `<version>` must be a single alphanumeric field, (e.g., 2.0rev0, 3.0beta3, or 3.0Betarev000); no spaces are allowed.
- Maximum length of `<version>` is 15 characters.

`Install` displays `<version>` on the MM selection screen to aid the operator in extension module selection.

vend:

Syntax: `vend: <vendor>`

Example: `vend: Cabletron`

Description: The `vend:` entry provides the Cabletron-assigned name of the vendor (`<vendor>`) creating the extension module.

- `<vendor>` may consist of several words.

With the exception of the fact that `mkmm` requires this entry, `vend:` is not used in the distribution and installation utilities. Its presence in the index file serves to identify the extension module vendor for software maintenance purposes.

Extension Module Relation Entries

Extension module relation entries describe relationships between the extension module associated with the index file and other extension modules.

comp:

Syntax: `comp: <mmname>`

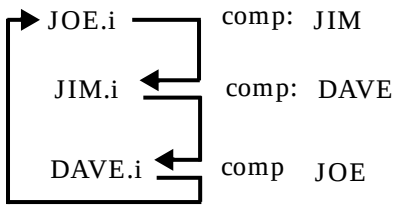
Example: `comp: CHASSIS MIMs`

Description: The `comp:` label enables developers to identify distribution (`mkdist`) dependencies between two or more SPECTRUM extension modules. A distribution dependency exists when the proper operation of an extension module relies upon the presence of one or more other software components. Each of these components must be distributed to the medium for the dependent component to work successfully. (Refer to the `dep:` discussion on page 3-12 for information about another type of dependency, pertaining to database relationships.)

Using `comp:` to indicate distribution dependencies between SPECTRUM extension modules ensures that all prerequisite components for a SPECTRUM extension are resident on the medium. This can reduce distribution-related problems at a customer site.

The `comp:` label indicates to `mkdist` that the extension module associated with the index file has a distribution dependency on the extension module(s) identified by `<mmname>`. For example, the `comp: JIM` entry in the index file `JOE.i` in the following diagram indicates to `mkdist` that extension module `JOE` has a distribution dependency on extension module `JIM`. If `JOE` and `JIM` exist in the same manufacturing area, `JOE` cannot be distributed to the medium without `JIM`.

- `comp:` allows only one dependency to be defined at a time. You can establish multiple dependencies by providing a sequence of `comp:` statements, each on its own line.
- `comp:` can be overridden by setting the `mkdist` "comp=n" option before invoking the `mkdist` tool. Using the example given above, for instance, if the `mkdist` "comp=n" option is set before `mkdist` is executed on `JOE.i`, extension module `JOE` will be distributed without extension module `JIM`.
- `comp:` dependencies can be circular. For example, assume the following lines in the index files for extension modules `JOE`, `JIM`, and `DAVE`:



The `mkdist` tool would distribute all three modules, because extension module `JOE` depends on extension module `JIM`, extension module `JIM` depends on extension module `DAVE`, and extension module `DAVE` depends on extension module `JOE`.

`comp:` cannot be used to establish a dependency between extension modules of different levels (refer to the `level:` discussion on page 3-7). If a file attempts to do this, `mkdist` will generate a warning message and ignore the dependency.

dep:

Syntax: `dep: <mmname>`

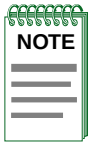
Example: `dep: SMSNP-CSI`

Description: The `dep:` label specifies a SpectroSERVER database dependency on another SPECTRUM extension module.

Where database files are packaged (**`db:`** or **`file: SS db`**), there must be a **`dep:`** entry.

The `dep:` label enables developers to identify any installation dependencies between two or more SPECTRUM extension modules to the `Install` program. An installation dependency exists when the proper operation of an extension relies upon the presence of one or more other software components. Each of these components must be installed on the SPECTRUM system for the dependent component to work successfully. (Refer to the `comp:` discussion on page 3-10 for information about another type of dependency, concerned with distribution relationships.)

Using `dep:` to indicate dependencies between SPECTRUM extension modules ensures that all prerequisite components for a SPECTRUM extension module are properly installed. This can reduce installation-related problems at a customer site.



The "dependencies" option of the `Install` *command enables you to list all known dependencies. For more information, refer to the* ***SPECTRUM Installation Guide****.*

The `dep:` label not only indicates to `Install` that a SpectroSERVER database dependency exists but also specifies the order of installation of dependent components. If the Cabletron DNI Management Module, `SMINT-CSI`, required that the Cabletron Management Module for the SNMP Protocol (`SMSNP-CSI`) be installed prior to installing `SMINT-CSI`, for example, then `SMINT-CSI.i`, the index file for `SMINT-CSI`, would contain the following entry:

```
dep: SMSNP-CSI
```

- When `Install` processes this index file entry, it makes sure that `SMSNP-CSI` is either on the distribution media and selected for installation, or is already installed on the system, before installing `SMINT-CSI`.
  - If `SMSNP-CSI` is currently installed, `Install` continues the installation process.
  - If `SMSNP-CSI` is not currently installed, but has been selected for installation from the distribution media, `Install` continues the installation process.
  - If `SMSNP-CSI` is not currently installed, and is not available on the distribution media, `Install` does not stop the installation but also does not install the dependent extension.
  - If `SMSNP-CSI` is not currently installed, and has not been selected for installation even though it is on the distribution media, `Install` displays a prompt informing the installer of the dependency and asks if `SMSNP-CSI` should be added to the list of options to be installed. If the installer decides not to install `SMSNP-CSI`, `Install` does not stop the installation but it also does not install the dependent extension.
- `dep:` allows only one dependency to be defined at a time. You can establish multiple dependencies by providing a sequence of `dep:` statements, each on its own line.
- `dep:` dependencies cannot be circular. For example, if hypothetical extension modules `A` and `B` had a mutual dependency and you put a `dep:` label in both `A` and `B`, that situation would constitute a circular dependency — to install `A`, you need to install `B` first; but to install `B`, you need to install `A` first; but …. If this condition occurs, `Install` displays an error message and discontinues the installation.
- The `dep:` label can establish a one-to-many dependency (to install `A`, first install `B` and `C`), or a chain of dependencies across several extension modules (to install `A`, first install `B` — but to install `B`, first install `C`).
- The `dep:` label can also establish an order of installation for multiple extension modules. For example, assume the following lines in the index files for extension modules `A`, `B`, and `C`:

```
AEXTM-WHO.i     dep:     BEXTM-WHO

BEXTM-WHO.i     dep:     CEXTM-WHO

CEXTM-WHO.i     dep:     DEXTM-WHO
```

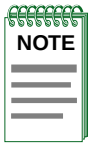
`Install` would install the modules in this order:

DEXTM-WHO
CEXTM-WHO
BEXTM-WHO
AEXTM-WHO

`Install` currently cannot manage `dep:` dependencies across multiple installation media. For this reason, developers must carefully plan the order of installation for dependent extension modules. The developer must ensure that a component on which other components depend either (a) resides on the same media as the module that depends on it, or (b) is installed prior to installing the module that depends on it.

Development planning is also important in the case of extension modules that have dependencies on program components that are originated by multiple vendors and distributed on different media.

If the installer attempts to install these components in the wrong order, `Install` alerts the installer that a particular component is uninstallable because a dependency cannot be met. The needed component is not already installed, for example, nor is it on the distribution media containing the module currently being installed. In this case, the installer has no alternative but to start over. To avoid this problem, the developer should determine the proper installation sequence and make sure that the order of installation is thoroughly tested and clearly documented.



During installation, SS database import files must be imported into the SS database in proper order. If your extension module distributes two or more SS database files, the **`file: SS db`** *entries for these files must appear in the index file in the same order in which they are to be imported.*

Also, if your extension module contains SS database import files, they will almost certainly have to be imported into the SS database after the SS database import files contained in some other extension module or extension modules. The **`dep:`** *entry is used to establish the import order of SS database files between different extension modules. If your extension module contains any SS database import files (for example, if the index file contains* **`file: SS db`** *entries), the index file must also contain at least one* `dep:` *entry. The* `mkmm` *tool automatically enforces this requirement.*

File Distribution Labels

File distribution labels are used to specify files to be included in an extension module, or to modify the way these extension module files are distributed or installed.

attr:

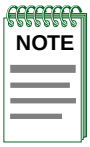
Syntax: `attr: <file> [mode=<mode>] [uid=root]`

Example: **`attr: SS/Spectroserver mode=4777 uid=root`**

Description: The `attr:` entry lets you change the accessibility of a file, effective following completion of installation.

Normally, after installation, installed files have the mode with which they were extracted from the medium or created in the installation area, and they are "owned" by the installation owner as specified via the `Install` GUI. However, there are times when it is desirable to override these default permissions or ownerships for specific files. The `attr:` entry can be used to accomplish this.

In the `attr:` entry, `<file>` is a file name, specified relative to the installation root directory, and `<mode>` is a UNIX **chmod**-style mode modifier. At the end of the installation, the mode modifier is applied to the specified file, as if a "**`chmod <mode> <file>`**" UNIX command had been executed. If the "uid=root" flag is also present in the `attr:` entry, `<file>` is given root ownership.



You can use **`file:`** *and* `attr:` *entries to enforce mode and ownership of installed files.*

Mode: *The mode of an extracted file is determined by the mode of the original file shipped and the "mode=<mode>" flag (if any) of the* **`file:`** *entry used to ship the file. Normally,* `Install` *does not change the mode of an installed file during installation, and it retains its extracted mode until the end of the installation. If* `Install` *does unacceptably modify a file's mode during installation, however, or if the file is created during installation (in which case it has no* **`file:`** *entry), its mode at the end of installation can be enforced by the use of an* `attr:` *entry*

Ownership: *Normally,* `Install` *changes ownership of installed files at the end of the installation to that of the installing user. The "uid=root" flag of the* `attr:` *entry is the only way to enforce root ownership of a file at the end of an installation. The* **`file:`** *entry has no flag to affect ownership of an installed file. Using* **`chown`** *to directly change ownership of a file in a custom script will not have the intended effect.*

deflt:

Syntax: `deflt: <prod> <type> <defname> <value> \ [plat=<plat>] [prot=yes] [req=yes]`

Example: **`deflt: SS xtn sdir /usr/swd/bin plat=sun4c`**

Description: The **`deflt:`** label associates a default value with a product and file type. These default values then become expanded within subsequent **`file:`** or **`head:`** entries that have matching product and file types.

`deflt:` entries are the only entries that may appear in the data files `mkmm.sys` and `mkmm.loc`. These **`deflt:`** entries establish the standard default values applied by `mkmm` and `mkmmdeps` to **`file:`** and **`head:`** entries in all index files. (For details on the use of **`deflt:`** entries in `mkmm.sys` and `mkmm.loc`, refer to the discussion of *mkmm.sys and mkmm.loc*, starting on page 4-11.)

In index files, **`deflt:`** entries are used to override the standard default values defined in `mkmm.sys` and `mkmm.loc`. The defaults set by a "**`deflt: <prod> <type> ...`**" entry in an index file apply to all subsequent **`file:`** and **`head:`** entries with the same `<prod>` and `<type>` fields as the **`deflt:`** entry, until the end of the index file or until another **`deflt:`** entry with the same `<prod>` and `<type>` fields is encountered.

<prod> File product

<type> File type

`<prod>` and `<type>` take on any combination of values legal for the `<prod>` and `<type>` fields of the **`file:`** entry. These values are described in detail in *Levels, Products and File Types*, starting on page 5-1 and in *Development of Shippable Files*, starting on page 5-3.

The entry "**`deflt: <prod> <type>...`**" specifies a default source or target directory value to be applied to subsequent **`file:`** and **`head:`** entries having the same `<prod>` and `<type>` fields.

<defname> Specifies the name of the default being assigned and may take on one or the other of the following two values:

`sdir`	Set the `<sdir>` default source directory.
`tdir`	Set the `<tdir>` default target directory.

<value> If `<defname>=sdir`, `<value>` may be an absolute or relative path name. If `<defname>=tdir`, `<value>` must

be a relative path name, relative to the current directory when installation is invoked. You can set <value> to the "?" variable to recover the system default value.

plat=<plat> This optional flag must occur after the <value> field. It specifies that the **deflt:** entry is to be processed only on platform <plat>. Multiple plat=<name> flags may be given to specify that the **deflt:** entry should be processed on all specified platforms.

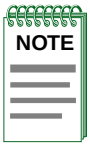
plat!=<plat> This optional field must occur after the <value> field. It specifies that the **deflt:** entry is to be processed only on platforms other than <plat>. Several plat!=<name> flags may be given on the same line, to indicate that the **deflt:** entry should not be processed on several platforms.

The **plat=plat** and **plat!=plat** flags are mutually exclusive; they cannot be used together in the same **deflt:** entry. If neither **plat=plat** nor **plat!=plat** flags are used, the **deflt:** entry is processed on every platform. The value of <plat> may be any of the standard platform names identified in Table 3-1, *Standard Platforms for SPECTRUM Extension Modules*, on page 3-6.

prot=yes If used, the optional **prot=yes** field must occur after the <value> field. Its presence specifies that files of prod <prod> and type <type> shipped via subsequent **file:** entries should be checksum-protected at installation time.

req=yes If used, this optional **req=yes** field must occur after the <value> field. Its presence specifies that this **deflt:** entry cannot be overridden by subsequent **deflt:** entries. VAR extension module developers should not use this option.

The **prot=yes** and **req=yes** options are intended for use only in the mkmm.sys data file and in Cabletron-produced index files. VAR extension module developers should not use these options.



The "**?**" fields in the following file: example (and elsewhere in these label descriptions) will be replaced by system default values for these identifications during processing by mkmm or mkmmdeps.

file:

Syntax:

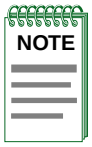
```
file: <prod> <type> <sdir> <sname> <tdir> \
<tname> [mode=<mode>] [uid=root] [prot=yes] \
[plat=<plat>]
```

Example:

```
file: SG iib ? CsIib/DecAgent/AgntDnArr.Bsc\
? ?<Machine_OUT>
```

Description:

The **`file:`** label specifies that a file should be distributed on a SPECTRUM distribution medium in a management module (MM) record. `mkmm` locates the source file in `<sdir>/<sname>`, and adds it to the extension module archive so that `Install` will later install it as `<tdir>/<tname>`, relative to the install root directory. The **`file:`** label provides complete flexibility in distributing files within an extension module.



On Windows NT platforms, if either `mkmm` *or* `mkmmdeps` *is unable to find a source file with the name specified in the index file, it will attempt to locate a file having the same name but with various common extensions (such as* `.exe`*,* `.cmd`*, and* `.bat`*). If* `mkmm` *or* `mkmmdeps` *find such a file, they add it to the extension module archive, using the applicable extension. This eliminates the need for platform-dependent index file entries to ship executable files on both Solaris and Windows NT. See Source File Extension Handling in mkmm and mkmmdeps, starting on page 4-13.*

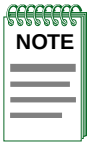
In the **`file:`** entry, the `<prod>`, `<type>`, `<sdir>`, `<sname>`, `<tdir>`, and `<tname>` fields are required, while the remaining fields are optional. Each field is described below:

`<prod>` File product.

`<type>` File type.

The `<prod>` and `<type>` fields respectively identify the product and file type associated with the file being distributed. The `mkmm` tool uses `<prod>` and `<type>` to help determine default directories for locating and installing the file (see `<sdir>` and `<tdir>` discussions, following). `Install` uses `<prod>` and `<type>` to decide what standard processing to apply to the file during installation.

For detailed information on acceptable combinations of values for <prod> and `<type>`, as well as on their effects on processing of shipped files by `mkmm` and `Install`, see Chapter 5, *Shippable Files*.



If **<prod>** = `Install`, *or* <**type**> = `cus`, *the file will be distributed, not in the MM record of the medium, but in the Install record (first record), as if the* `head:` *label had been used. In VAR-developed extension modules, the only files that normally need to be shipped in the Install record are MM description files and custom scripts.*

<sdir> — <sdir> is the source directory of the file to be distributed. <sdir> may be an absolute path name, a relative path name (relative to the directory where mkmm is invoked), or the "?" variable (which causes mkmm to supply a system default value). This supplied value either is a standard default value depending on <prod> and <type>, or else is a default value defined in a previous **deflt:** command. For detailed information on system defaults for <sdir>, see Table 5-3, *File Statistics by Product and Type*, on page 5-6. (For more information on **deflt:**, see the **deflt:** discussionon page 3-16.)

<sname> — <sname> is the source name of the file to be distributed. <sname> must be an explicitly relative path name or file base name (relative to <sdir>). If <sname> is expressed as the "?" variable, however, mkmm will fail.

<tdir> — <tdir> is the target directory of the file when it is installed. <tdir> may be either a relative path name (relative to the installation root directory) or the "?" variable. If <tdir> is the "?" variable, mkmm supplies a default value. This value is either a standard default value, depending on <prod> and <type>, or a default value defined in a previous **deflt:** command. For detailed information on system defaults for <tdir>, see Table 5-3, *File Statistics by Product and Type*, on page 5-6. (For more information on **deflt:**, see the **deflt:** discussion on page 3-16.)

<tname> — <tname> is the target name of the file when it is installed. <tname> may be any of a relative path name, or a file basename (relative to <tdir>), or the "?" variable. If <tdir> is the "?" variable, mkmm defaults that field to the value of <sname>.

mode=<mode> — The optional file mode field must occur after the <tname> field. It specifies that the POSIX command "**chmod <mode>**" must be applied to the file before committing it to the extension module archive file. The permissions of the source file in the development area remain unchanged. `Install` later extracts the file from the medium with the modified permissions. The value given for <mode> must be a valid mode change description argument for the POSIX chmod utility. If unspecified, the source file is committed to

the extension module archive file and installed with the existing permissions.

For more information about enforcing mode and ownership of installed files, see the **attr:** entry on page 3-15.

uid=root The optional file owner field must occur after the <tname> field. It specifies that the command "**chown root**" must be applied to the file before committing it to the extension module archive file. The ownership of the source file in the development area remains unchanged. Install later extracts the file from the medium with root ownership. If unspecified, the source file is committed to the extension module archive file and installed with the ownership of the installing user.

For more information about enforcing mode and ownership of installed files, see the **attr:** entry on page 3-15.

prot=yes If used, this optional field must occur after the <tname> field. Its presence specifies that the shipped file should be checksum-protected at installation time.

The **prot=yes** option is intended for use only in Cabletron-produced index files. VAR extension module developers should not use this option.

plat=<plat> This optional flag must occur after the <tname> field. This flag specifies that the **file:** entry is to be processed only on the platform designated by the <plat> field. (Multiple plat=<name> flags may be given in sequence on the same line, however, so as to specify that the **file:** entry can be processed on any or all of the specified platforms.)

plat!=<plat> This optional field must occur after the <tname> field. This flag specifies that the **file:** entry is to be processed only on platforms other than the one designated by the <plat> field. (Again, several plat!=<name> flags may be given on the same line, thereby indicating that the **file:** entry should not be processed on any of the designated several platforms.)

plat=plat and plat!=plat flags cannot be used together in the same **file:** entry. If neither plat=plat nor plat!=plat flags are used, the **file:** entry is processed on every platform. <plat> may be any of the standard platform names identified in Table 3-1, *Standard Platforms for SPECTRUM Extension Modules*, on page 3-6.

fdir:

Syntax: `fdir: <prod> <type> <sdir> <snamepat> \ <tdir> <tname> [mode=<mode>] [uid=root] \ [prot=yes] [plat=<plat>]`

Example: **`fdir: SG iib ? \.csi$ ? ?`**

Description: The **`fdir:`** entry specifies that `mkmm` should traverse a directory, locating all files having base names (filenames, less any extension suffix) that match a specified pattern, and generate **`file:`** entries to ship all matching files thus found.

The **`fdir:`** entry is useful for shipping all the files of a given type (identifiable by base name) within a specified directory, where the directory structure of the shipped files should correspond to the directory structure of the files in the development area.

In the **`fdir:`** entry, the `<prod>`, `<type>`, `<sdir>`, `<snamepat>`, `<tdir>`, and `<tname>` fields are required, while the remaining fields are optional.

When `mkmm` processes an **`fdir:`** entry, it traverses the `<sdir>` directory looking for files to add to the extension module archive. Whenever it finds a file having a base name that matches the pattern `<snamepat>`, it generates a **`file:`** entry to ship the matching file. The `<sname>` and field of each generated **`file:`** entry is the name of a matched file, relative to `<sdir>`. The `<tname>` field of each **`file:`** entry is ? (regardless of the `<tname>` specified in the **`fdir:`** entry). All other fields of the generated **`file:`** entries, including the `<prod>`, `<type>`, `<sdir>`, `<tdir>`, and flag fields, are the same as the corresponding fields of the **`fdir:`** entry.

The `<snamepat>` pattern field must be a regular expression, as used by the `Perl` programming language (similar to the regular expressions used by the UNIX **`grep`** command). `<snamepat>` is used to match the *base names* of files encountered during traversal of the `<sdir>` directory, not their *path names* relative to `<sdir>`. While this document does not describe the complete syntax of `Perl`-style regular expressions, the examples in Table 3-2 of files that are matched by specific `<snamepat>` patterns should help with respect to showing how to compose most **`fdir:`** entries:

Table 3-2. Significance of Basic <snamepat> Patterns

<snamepat>	Matches files whose base name ...
\.csi$	ends with extension **.csi**
^[a-z]	begins with a lower case letter

Table 3-2. Significance of Basic <snamepat> Patterns (Continued)

<snamepat>	Matches files whose base name ...
^Makefile$	is "Makefile"
file	contains the substring "file"

head:

Syntax: `head: <prod> <type> <sdir> <sname> <tdir> \ <tname> [plat=<plat>] [mode=<mode>]`

Description: The **`head:`** label specifies that a file should be distributed in the Install record (first record) of the medium. In every other respect, including syntax, the **`head:`** entry is precisely the same as the **`file:`** label. Any **`file:`** entry with `<prod> = Install` or `<type> = cus` is interpreted as a **`head:`** entry.

The `head:` *label should be used only to ship files that must exist on the installation system before* `Install` *is invoked (that is, to ship utilities that* `Install` *uses). This label should not be used in VAR-developed index files.*

plat:

Syntax: `plat: <platlist>`

Example: **`plat: intel sun4c`**

Description: The **`plat:`** entry is used to specify the set of SPECTRUM installation platforms on which the extension module may be successfully built and shipped. The `<platlist>` field is a list of one or more standard index file platform names, as identified in Table 3-1, *Standard Platforms for SPECTRUM Extension Modules*, on page 3-6.

If an index file contains any entries with "plat=" or "plat!=" modifiers, the **`plat:`** entry must appear in the index file before any of these entries. If an index file contains no entries with "plat=" or "plat!=" modifiers, the **`plat:`** entry may be omitted.

If an index file contains a **`plat:`** entry, and an attempt is made to build an extension module archive on a platform not listed in that **`plat:`** entry, `mkmm` will fail.

Cabletron-Reserved Labels

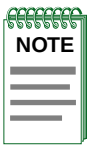
The following labels are reserved for Cabletron and should not be used in VAR-developed index files.

Cabletron-Reserved labels are restricted for use by Cabletron and are subject to undocumented changes. Unauthorized use of these Cabletron Reserved labels may produce unexpected results if such usage happens to allow data exchange beyond your intended limitations of program specifications.

info: ***mtype:***

Index File Entry Examples

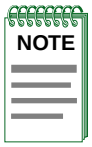
This section contains numerous examples of new-style index file entries. These examples are intended to provide you with "how to" type information that will make writing your own index files easier. Figure 3-1 delineates the entries in `SMACME-MM.i`, a fictitious index file describing the components of extension module `SMACME-MM`. The entries in this file will serve as examples for the following discussion. For the purpose of illustrating how this index file is used, this discussion assumes the development area for the extension module to be `/usr/ACME/devarea/buildfiles/SMACME-MM.i` and assumes the SEI Toolkit to be installed in `/usr/Spectrum/INSDK`.



Because of the character-count limitation of the page widths in this publication, foldover lines in reproduced or simulated program coding used as examples in this document include the backslash character to show folding has occurred in accordance with standard Unix convention, for the sake of clarity of explanation. In an actual index file, backslashes cannot be used to continue an entry across more than one line (which may contain up to 255 characters).

Figure 3-1. Entries in the SMACME-MM.i Index File (Fictitious)

```
 1   mm:  ACME Management Module
 2   vend:ACME
 3   rev: 1.0rev0
 4   irev:01.00.00.000
 5   level:1
 6   plat:sun4c
 7   dep: SMEPI-CSI
 8   comp:SNMP_Device
 9   file:Install mmdesc SMACME-MM.mmd ? ?
10   file:SG iib ${DEVAREA}}/SG-Support \
     CsIib/DecAgent/AgntDnArr.Bsc ? ?
11   file:SS db ${DEVAREA}/dbfiles IRBM.e ? ?
12   file:SS ih ${DEVAREA}/src/intel CSIHDecnet.o ? ?
13   file:SS cus ? ACME.cus ? ?
14   file:SS doc ${DEVAREA}/docfiles/doc.nt ? \
     ACME.readme plat=ibm
15   deflt:SS file sdir /usr1/swd/kinet/bin
16   deflt:SS file tdir ACME
17   file:SS file ? util1 ? ?
18   file:SS file ? util2 ? ?
19   deflt:SS file sdir ?
20   deflt:SS file tdir ?
21   deflt:SG file sdir ${DEVAREA}/src
22   deflt:SG file sdir ${DEVAREA}/src.nt plat=ibm
23   deflt:SG file tdir ACME
24   file:SG file ? util3 ? ?
25   file:SG file ? util4 ? ?
26   deflt:SG file sdir ?
27   deflt:SG file tdir ?
28   fdir: SG iib /user/devarea/CsIib/Bases \.Bas$ \
     SG-Support/CsIib/Bases ? mode=444
```



The braces used with ${DEVAREA} in this example code are not actually necessary (refer to the Environment Variables discussion on page 3-2), but developers would be wise to adopt this habitual caution so as to ensure correct expansion of variables in the index files.

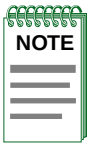
Having developed such an index file, we can now proceed to the manufacturing phase, using the following Bourne shell commands:

```
cd /usr/ACME/devarea/buildfiles
DEVAREA=/usr/ACME/devarea
export DEVAREA
/usr/Spectrum/INSDK/mkmm SMACME-MM.i
```

Upon execution of the `/usr/Spectrum/INSDK/mkmm SMACME-MM.i` command at the end of this sequence, `mkmm` will attempt to build `SMACME-MM.tar` from `SMACME-MM.i`. In order for `mkmm` to be successful, all index file syntax rules must be observed. To this end, each of the following

examples provides syntactical and descriptive information for the label being discussed, as well as a synopsis of how mkmm processes the label. For detailed information about a particular label, refer to *Index File Entry Definitions*, starting on page 3-6.

The following examples are arranged by functional group: first Extension Module Description and then File Distribution. In some cases, where entries interact, groups of entries are discussed as a unit.



For information on how mkmm *processes old-style File Distribution entries, refer to Index File Conversion, starting on page 3-37.*

Extension Module Description Label Examples

The following examples of Extension Module Description label entries are from SMACME-MM, the fictitious file described in the previous subsection (see Figure 3-1). These labels are representative of Extension Module Description labels for any index file. All Extension Module Description entries should appear in every index file; omission of an Extension Module Distribution entry will not inhibit mkmm from processing the index file, but it may cause unexpected errors during distribution and/or installation of the extension module. For detailed information about specific Extension Module Description labels, refer to *Index File Entry Definitions*, starting on page 3-6.

Example Line 1 – mm:

Entry: **mm: ACME Management Module**

In this example, the <name> identification of the module is "ACME Management Module." mkmm moves this entry to the SMACME-MM archive file unaltered.

Example Line 2 – vend:

Entry: **vend: ACME**

In this example, the <vendor> name is "ACME." mkmm moves this entry to the SMACME-MM archive file unaltered.

Example Line 3 – rev:

Entry: `rev: 1.0rev0`

In this example, the human-friendly `<version>` identification of the module is "1.0rev0." `mkmm` moves this entry to the `SMACME-MM` archive file unaltered.

Example Line 4 – irev:

Entry: `irev: 01.00.00.000`

In this example, the machine-friendly `<version>` identification of the module is "01.00.00.000." `mkmm` moves this entry to the `SMACME-MM` archive file unaltered.

Example Line 5 – level:

Entry: `level: 1`

In this example, the `<level>` identification of the module is "1." `mkmm` moves this entry to the `SMACME-MM` archive file unaltered.

Example Line 6 – plat:

Entry: `plat: sun4c`

In this example, `SMACME-MM` has applicability only to the sun4c platform, as defined in Table 3-1. This means that the `SMACME-MM` can be installed on SUN Solaris machines, but not on Windows NT.

File Distribution Entry Examples

The following examples are representative of File Distribution label entries. File Distribution labels affect the way files are distributed. Refer to the File Distribution Label section of this chapter for detailed information about File Distribution labels. `mkmm` fills in default values, expands environment variables, and then moves the expanded entry to the extension module archive file (`MM.tar`) associated with the extension module being distributed—in this case, the fictitious `SMACME-MM` file described in the preceding subsection.

Refer to *Development: mkmm*, starting on page 4-7, for more information about the mkmm tool.

Example Line 7 – dep:

Entry: **dep: SMEPI-CSI**

In this example, SMACME-MM has a database dependency on extension module SMEPI-CSI. This means that the database files associated with extension module SMEPI-CSI must be imported into the SPECTRUM database before the SPECTRUM database files associated with SMACME-MM are installed. mkmm moves this entry to the extension module archive file unaltered.

Example Line 8 – comp:

Entry: **comp: SNMP_Device**

In this example, SMACME-MM has a mkdist distribution dependency on SNMP_Device. This means that, during mkdist distribution, the files associated with SNMP_Device must be distributed to the medium along with the files associated with SMACME-MM. mkmm moves this entry to the extension module archive file unaltered.

Example Line 9 – file: Install mmdesc

Entry: **Install mmdesc? SMACME-ME.mmd ? ?**

In this entry, the Basename.mmd identification must be the same as the name of the **.i** file (e.g., SMACME-ME.mmd). If the index file were named SM-CSI1000.i, for example, the applicable Basename.mm entry would be SM-CSI1000.mmd. The .mmd file is a short clear-text description of the module. If this entry is missing during the installation process, no description will be visible on the component selector window during the install process.

Example Line 10 – file: SG iib

Entry:

```
SG iib ${DEVAREA}/SG-Support \
CsIib/DecAgent/AgntDnArr.Bsc ? ?
```

In this example:

<prod> = **SG**
<type> = **iib**
<sdir> = **${DEVAREA}/SG-Support**
<sname> = **CsIib/DecAgent/AgntDnArr.Bsc**
<tdir> = **?**
<tname> = **?**

This entry tells mkmm that a SpectroGRAPH Iib (Icon information block) file is being included in the extension module archive. Since the <tdir> and <tname> fields of the entry both have "?" default values, mkmm will attempt to substitute system default values for these fields.

Since the file is of product SG and type iib, mkmm sets <tdir> to SG-Support (see Table 5-3, *File Statistics by Product and Type*, on page 5-6). The "?" variable in the <tname> field defaults to <sname>, which in this case is CsIib/DecAgent/AgntDnArr.Bsc. When all the defaults are expanded, the entry becomes:

```
file: SG iib ${DEVAREA}/SG-Support \
CsIib/DecAgent/AgntDnArr.Bsc SG-Support
CsIib/DecAgent/AgntDnArr.Bsc
```

Next, mkmm sees the Bourne shell environment variable ${DEVAREA}[1] and expands it to its original value, as set and exported before mkmm was invoked. This value is /usr/ACME/devarea. (If ${ext}[1] had not been set and exported before mkmm was invoked, an error would have occurred.) When the environment variable is expanded, the entry becomes:

```
file: SG iib /usr/ACME/devarea/SG-Support \
CsIib/DecAgent/AgntDnArr.Bsc SG-Support\
CsIib/DecAgent/AgntDnArr.Bsc
```

The mkmm utility thus will locate the source file at /usr/ACME/devarea/SG-Support/CsIib/DecAgent/

[1] The braces used with ${DEVAREA} in this example (and with ${ext}, below) are not actually necessary (refer to the *Environment Variables* discussion on page 3-2), but developers would be wise to adopt this habitual caution so as to ensure correct expansion of variables in the index files.

`AgntDnArr.Bsc` and place it in the extension module archive file, so that it will later become installed at `SG-Support/CsIib/DecAgent/AgntDnArr.Bsc`.

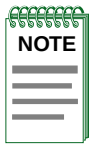
Example Line 11– file: SS db

Entry: **`file: SS db ${DEVAREA}/dbfiles IRBM.e ? ?`**

In this example:

`<prod>` = **SS**
`<type>` = **db**
`<sdir>` = **${DEVAREA}/dbfiles**
`<sname>` = **IRBM.e**
`<tdir>` = **?**
`<tname>` = **?**

This entry tells `mkmm` that a SpectroSERVER database import file is being included in the extension module archive.



The `<sdir>` *field is given explicitly. If* `<sdir>` *were given as the "?" variable,* `mkmm` *would attempt to set* `<sdir>` *to the standard default* `<sdir>` *value for files of product* SS *and type* db *(refer to Table 5-3, File Statistics by Product and Type, on page 5-6). This is a Cabletron-specific path; it would not typically exist in other development environments. There are at least two ways to handle this situation. One way is to give an explicit* `<sdir>` *field in the* **`file:`** *entry, as was done in this example. Another way is to use a* **`deflt:`** *entry to change the default* `<sdir>` *value for* **`file: SS db`** *entries. This latter solution would be preferred if there were a number of* **`file: SS db`** *entries in the index file.*

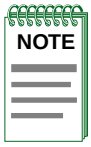
`<tdir>` and `<tname>` are both set for the "?" variable, so `mkmm` supplies system default values for them. In this case, `<tdir>` defaults to SS (refer to Table 5-3), and `<tname>` defaults to the value of `<sname>`, which is `IRBM.e`. With these default values expanded, the entry becomes:

```
file: SS db ${DEVAREA}/dbfiles IRBM.e SS IRBM.e
```

Next, `mkmm` expands the Bourne shell environment variable `${DEVAREA}` to its original value, as set and exported before `mkmm` was invoked. In this example, the value of `${DEVAREA}` is `usr/ACME/devarea`. With the Bourne Shell environment variable expanded, the entry becomes:

```
file: SS db /usr/ACME/devarea/dbfiles \
IRBM.e SS IRBM.e
```

`mkmm` will locate the source SpectroSERVER database import file at `/usr/ACME/devarea/dbfiles/IRBM.e` and then will add it to the extension module archive—so that it will later install at `SS/IRBM.e` relative to the installation root directory.



During installation, SS database import files must be imported into the SS database in proper order. If your MM distributes two or more SS database files, the **`file: SS db`** *entries for these files must appear in the index file in the same order in which they are to be imported. Also, if your MM contains SS database import files, they will almost certainly have to be imported into the SS database after the SS database import files contained in some other MM or MMs. The* **`dep:`** *entry is used to establish the import order of SS database files between different MMs. If your MM contains any SS database import files (if the index file contains* **`file: SS db`** *entries, for example), the index file must also contain at least one* **`dep:`** *entry.* `mkmm` *automatically enforces this requirement.*

Example Line 12 – file: SS ih

Entry:

```
file: SS ih ${DEVAREA}/src/intel \
CSIHDecnet.o ? ?
```

In this example:

`<prod>` = **SS**
`<type>` = **ih**
`<sdir >` = **${DEVAREA}/src/intel**
`<sname>` = **CSIHDecnet.o**
`<tdir>` =**?**
`<tname>` = **?**

This entry tells `mkmm` that a SpectroSERVER `ih` (inference handler) file is being included in the extension module archive. The "?" variable is entered in the `<tdir>` and `<tname>` fields of the entry, so `mkmm` will attempt to substitute system default values for these fields.

Since the file is of product SS and type `ih`, `mkmm` sets `<tdir>` to **SS** (refer to Table 5-3). `<tname>` defaults to `<sname>`, which in this case is `CSIHDecnet.o`. When all these defaults are expanded, the entry becomes:

```
file: SS ih ${DEVAREA}/src/intel \
CSIHDecnet.o SS CSIHDecnet.o
```

Next, mkmm sees the Bourne shell environment variable ${DEVAREA} and expands it to its original value, as set and exported before mkmm was invoked. This value is /usr/ACME/devarea. (If ${DEVAREA} had not been set and exported before mkmm was invoked, an error would have occurred.) When the environment variable is expanded, the entry becomes:

```
file: SS ih /usr/ACME/devarea/src/intel \
CSIHDecnet.o SS CSIHDecnet.o
```

mkmm thus will locate the source file at /usr/ACME/devarea/src/intel/ CSIHDecnet.o and place it in the extension module archive file, so that it will later install at SS/CSIHDecnet.o.

Example Line 13 – file: SS cus

Entry: **file: SS cus ? ACME.cus ? ?**

In this example:

<prod> = **SS**
<type> = **cus**
<sdir> = **?**
<sname> = **ACME.cus**
<tdir> = **?**
<tname> = **?**

This entry tells mkmm that a SpectroSERVER cus (custom script) file is being included in the extension module archive. Since the "?" variable is entered in the <sdir>, <tdir>, and <tname> fields of the entry, mkmm will attempt to substitute system default values for these fields.

Since the file is of product SS and type cus, mkmm sets both <sdir> and <tdir> to "." (see Table 5-3). <tname> defaults to <sname>, which in this case is ACME.cus. When all these defaults are expanded, the entry becomes:

```
file: SS cus . ACME.cus . ACME.cus
```

Also, since the file is of type cus, this entry is interpreted as a **head:** entry. The entry then becomes:

```
head: SS cus . ACME.cus . ACME.cus
```

mkmm thus will locate the source file at ./ACME.cus and place it in the extension module archive file, so that it will later install on the Install record (the first record on the medium) at ./ACME.cus.

Example Line 14 – file: SS doc

Entry:

```
file: SS doc ${DEVAREA}/docfiles doc.nt ? \
ACME.readme plat=ibm
```

In this example:

<prod> = **SS**
<type> = **doc**
<sdir> = **${DEVAREA}/docfiles**
<sname> = **doc.nt**
<tdir> = **?**
<tname> = **ACME.readme**
plat=<name> = **plat=ibm**

This entry tells mkmm that a SpectroSERVER doc (document) file is being included in the extension module archive. The <tdir> field of the entry is specified as the "?" variable, so mkmm will attempt to substitute a system default value for this field.

Since the file is of product SS and type doc, mkmm sets <tdir> to Readme (refer to Table 5-3). When <tdir> is expanded, the entry becomes:

```
file: SS doc ${DEVAREA}/docfiles doc.nt \
Readme ACME.readme plat=ibm
```

Next, mkmm sees the Bourne shell environment variable ${DEVAREA} and expands it to its original value, as set and exported before mkmm was invoked. This value is /usr/ACME/devarea. (If ${DEVAREA} had not been set and exported before mkmm was invoked, an error would have occurred.) When the environment variable is expanded, the entry becomes:

```
file: SS doc /usr/ACME/devarea/docfiles \
doc.nt Readme ACME.readme plat=ibm
```

If this file is distributed to a Windows NT medium, therefore, mkmm will locate the source file at /usr/ACME/devarea/docfiles/doc.nt and place it in the extension module archive file, so that it will later install at Readme/ACME.readme. If the distribution is to a non-NT medium, however, the mutually exclusive setup means that this file will not be distributed to that medium. For more information, refer to the platform-identification specification discussion on page 3-20

Example Lines 15–20 – deflt:

Syntax: `deflt: <prod> <type> <defname> <value> plat=<name>`

Description: The **deflt:** label is used to associate a default value with a product and file type. This <value> will be applied to all subsequent entries with the same <prod> and <type>.

Entry:

```
deflt:SS file sdir /usr1/swd/kinet/bin
deflt:SS file tdir ACME
file:SS file ? util1 ? ?
file:SS file ? util2 ? ?
deflt:SS file sdir ?
deflt:SS file tdir ?
```

The first two lines of this sample entry establish <sdir> and <tdir> defaults for the two **file:** entries that follow.

In the first line:

<prod> = **SS**
<type> = **file**
<defname> = **sdir**
<value> = **/usr1/swd/kinet/bin**

This entry tells mkmm to apply the <sdir> value **/usr1/swd/kinet/bin** to all subsequent entries of product SS and type file.

The second line tells mkmm to apply the <tdir> value ACME to all subsequent entries of product SS and type file.

Since the "?" variable is given for the <sdir>, <tdir>, and <tname> fields of the **file:** entries, mkmm attempts to substitute system default values for these fields. Because

the entries are of product SS and type file, however, the default values established for <sdir> and <tdir> by the first two **deflt:** entries will be applied. <tname> defaults to <sname>, which in this case is respectively util1 and util2 for the two file entries.

When all the defaults are expanded, therefore, the **file:** entries become:

```
file: SS file /usr1/swd/kinet/bin util1 ACME util1
file: SS file /usr1/swd/kinet/bin util2 ACME util2
```

Therefore, mkmm will locate the source files at /usr1/swd/kinet/bin/util1 and /usr1/swd/kinet/bin/util2 and place them into the extension module archive file, so that they will later install as ACME/util1 and ACME/util2.

The last two entries restore the <sdir> and <tdir> values back to the system default values associated with <prod> = **SS** and <type> = **file**.

Example Lines 21–27 – deflt:

Entry:

```
deflt:SG file sdir ${DEVAREA}/src
deflt:SG file sdir ${DEVAREA}/src.nt \
plat=ibm
deflt:SG file tdir ACME
file:SG file ? util3 ? ?
file:SG file ? util4 ? ?
deflt:SG file sdir ?
deflt:SG file tdir ?
```

The first three lines of this example establish <sdir> and <tdir> defaults for the two **file:** entries that follow.

In the first line:

<prod> = **SG**
<type> = **file**
<defname> = **sdir**
<value> = **${DEVAREA}/src**

This entry tells mkmm to apply the <sdir> value ${DEVAREA}/src to all subsequent entries of product SG and type file for all platforms.

The second line tells mkmm to apply the <sdir> value ${DEVAREA}/src.nt to all subsequent entries of product

SG and type `file` for Windows NT platforms only. This command overrides the default set on the first line.

The third line tells `mkmm` to apply the `<tdir>` value ACME to all subsequent entries of product SG and to type `file` for all platforms.

Since the "?" variable is given for the `<sdir>`, `<tdir>`, and `<tname>` fields of the **`file:`** entries, `mkmm` attempts to substitute system default values for these fields. Because the entries are of product SG and type `file`, however, the default values established for `<sdir>` and `<tdir>` by the first three **`deflt:`** entries will be applied. `<tname>` defaults to `<sname>`, which in this case is respectively **`util3`** and **`util4`** for these two files. When all the defaults are expanded for non-NT platforms, the **`file:`** entries become:

```
file: SS file ${DEVAREA}/src util3 ACME util3
file: SS file ${DEVAREA}/src util4 ACME util4
```

Next, `mkmm` sees the Bourne shell environment variable `${DEVAREA}` and expands it to its original value, as set and exported before `mkmm` was invoked. This value is `/usr/ACME/devarea`. (If `${DEVAREA}` had not been set and exported before `mkmm` was invoked, an error would have occurred.) When this environment variable is expanded, therefore, the entries become:

```
file: SS file /usr/ACME/devarea/src \
util3 ACME util3
file: SS file /usr/ACME/devarea/src \
util4 ACME util4
```

`mkmm` thus will locate the source files at `/usr/ACME/devarea/src/util3` and `/usr/ACME/devarea/src/util4` and place them into the extension module archive file, so that they will later install at `ACME/util3` and `ACME/util4`.

When all the defaults are expanded for a Windows NT platform, the **`file:`** entries become:

```
file: SS file ${DEVAREA}/src.nt util3 ACME util3
file: SS file ${DEVAREA}/src.nt util4 ACME util4
```

Next, `mkmm` sees the Bourne shell environment variable `${DEVAREA}` and expands it to its original value, as set and exported before `mkmm` was invoked. This value is

"/usr/ACME/devarea." (If ${DEVAREA} had not been set and exported before mkmm was invoked, an error would have occurred.) When this environment variable is expanded, the entries become:

```
file: SS file /usr/ACME/devarea/src.nt \
util3 ACME util3
file: SS file /usr/ACME/devarea/src.nt \
util4 ACME util4
```

mkmm thus will locate the source files at /usr/ACME/devarea/src.nt/util3 and /usr/ACME/devarea/src.nt/util4 and place them into the extension module archive file, so that they will later install at ACME/util3 and ACME/util4.

The next two entries restore the <sdir> and <tdir> values back to the system default values associated with **<prod> = SS** and **<type> = file**.

Example Line 28– file: SS db

Entry: **fdir: SG iib /user/devarea/CsIib/Bas \.Bas$ \ SG-Support/CsIib/Bases ? mode=444**

Suppose you had a development area containing the following files:

```
/user/devarea/CsIib/Bases/Large/CT_TRHub.Bas
/user/devarea/CsIib/Bases/Large/NetAn.Bas
/user/devarea/CsIib/Bases/Large/Rx4820C.Bas
/user/devarea/CsIib/Bases/Med/Router.OPR
/user/devarea/CsIib/Bases/Small/NA_SGI_NV.Bas
/user/devarea/CsIib/Bases/Small/NetAnMIM.OPR
/user/devarea/CsIib/Bases/Small/Rx2244.OPR
/user/devarea/CsIib/Bases/Small/Rx4660S.OPR
/user/devarea/CsIib/Bases/Small/SNMP_Dev.OPR
/user/devarea/CsIib/Bases/Small/WA_Seg.Bas
/user/devarea/CsIib/Bases/Small/XY.Bas
```

Suppose further that you wished to ship all those files in the directory /user/devarea/CsIib/Bases that had the extension .Bas as SpectroGRAPH IIB files, maintaining their relative directory structure. To accomplish this, you might use the following **fdir:** entry:

```
fdir: SG iib /user/devarea/CsIib/Bases \.Bas$ \
SG-Support/CsIib/Bases ? mode=444
```

Because <sdir> = /user/devarea/CsIib/Bases, mkmm traverses the /user/devarea/CsIib/Bases directory. Similarly, because <snamepat> = \.Bas$, mkmm looks for files having base names that end in .Bas. In this case, the matching files it finds are as follows:

```
/user/devarea/CsIib/Bases/Large/CT_TRHub.Bas
/user/devarea/CsIib/Bases/Large/NetAn.Bas
/user/devarea/CsIib/Bases/Large/Rx4820C.Bas
/user/devarea/CsIib/Bases/Small/NA_SGI_NV.Bas
/user/devarea/CsIib/Bases/Small/WA_Seg.Bas
/user/devarea/CsIib/Bases/Small/XY.Bas
```

As a result, mkmm generates the following **file:** entries:

```
file: SG iib /user/devarea/CsIib/Bases Large/CT_TRHub.Bas\
SG-Support/CsIib/Bases ? mode=444
file: SG iib /user/devarea/CsIib/Bases Large/NetAn.Bas \
SG-Support/CsIib/Bases ? mode=444
file: SG iib /user/devarea/CsIib/Bases Large/Rx4820C.Bas \
SG-Support/CsIib/Bases ? mode=444
file: SG iib /user/devarea/CsIib/Bases
Large/NA_SGI_NV.Bas\
SG-Support/CsIib/Bases ? mode=444
file: SG iib /user/devarea/CsIib/Bases Large/WA_Seg.Bas \
SG-Support/CsIib/Bases ? mode=444
file: SG iib /user/devarea/CsIib/Bases Large/XY.Bas \
SG-Support/CsIib/Bases ? mode=444
```

Note that the <prod>, <type>, <sdir>, <tdir>, and flag fields (in this case mode=444) have propagated without change from the **fdir:** entry to each of the **file:** entries. Since these **file:** entries replace the original **fdir:** entry in the index file, they are thereafter processed as if they had existed in the original index file.

Index File Conversion

Introduction

As of SPECTRUM 4.0, mkmm no longer accepts some of the old-style entries in index files when building extension module archives. Therefore, you must convert any pre-existing index files to SPECTRUM 4.x/5.0 format before using mkmm. You can convert index files from old-style to new-style either manually or automatically (using mkmmconv). The following subsections explain both approaches.

Manual Index File Conversion

For ease and simplicity, we recommend that you convert index files automatically, using `mkmmconv`. If you prefer to convert your index file manually, however, you can accomplish this by replacing each old-style entry in the index file with a corresponding new-style entry, as identified in Table 3-3.

Table 3-3. Old-Style Index File Entries and New-Style Equivalents

Old-style Entry	New-style Entry
`comp: <mmname> …`	`comp: <mmname> …`
`cus_sg: <sdir>/<sname>`	`file: SG cus <sdir> <sname> ? ?`
`cus_sg: <sname>`	`file: SG cus ? <sname> ? ?`
`cus_ss: file:<sdir>/<sname>`	`file: SS cus <sdir> <sname> ? ?`
`cus_ss: <sname>`	`file: SS cus ? <sname> ? ?`
`db: <sdir>/<sname>`	`file: SS db <sdir> <sname> ? ?`
`db: <sname>`	`file: SS db ? <sname> ? ?`
`dep: <mmname> …`	`dep: <mmname> …`
`gib: <sname>`	`file: SG gib ? <sname> ? ?`
`icon: <sdir>/<sname>`	`file: SG icon <sdir> <sname> ? ?`
`icon: <sname>`	`file: SG icon ? <sname> ? ?`
`ih: <sdir>/<sname>`	`file: SS ih <sdir> <sname> ? ?`
`ih: <sname>`	`file: SS ih ? <sname> ? ?`
`iib: <sname>`	`file: SG iib ? <sname> ? ?`
`image: <sname>`	`file: SG image ? <sname> ? ?`
`irev: <version>`	`irev: <version>`
`level: <level>`	`level: <level>`
`mi: <sdir>/<sname>`	`file: SS mi <sdir> <sname> ? ?`
`mi: <sname>`	`file: SS mi ? <sname> ? ?`
`mm: <name>`	`mm: <name>`
`others: <sname>`	`file: SG other ? <sname> ? ?`
`pib: <pibfile> <pibline>`	`pib:` entries must be removed from the index file, converted, and shipped in a PIB contribution file. Note that the `mkmmconv` utility will convert `pib:` entries for you. For more information, refer to *PIB Contribution Files*, starting on page 5-30.
`rev: <version>`	`rev: <version>`
`rib: <sname>`	`file: SG rib ? <sname> ? ?`
`sg_obj: <sdir>/<sname>`	`file: SG obj <sdir> <sname> ? ?`
`sg_obj: <sname>`	`file: SG obj ? <sname> ? ?`
`ss_obj: <sdir>/<sname>`	`file: SS obj <sdir> <sname> ? ?`

Table 3-3. Old-Style Index File Entries and New-Style Equivalents (Continued)

Old-style Entry	New-style Entry
ss_obj: <sname>	file: SS obj ? <sname> ? ?
vend: <vendor>	vend: <vendor>
vfile: <sname>	file: SG vfile ? <sname> ? ?
view: <sdir>/<sname>	file: SG view <sdir> <sname> ? ?
view: <sname>	file: SG view ? <sname> ? ?
xtn_sg: <sdir> <sname>	file: SG xtn <sdir> <sname> ? ?
xtn_ss: <sdir> <sname>	file: SS xtn <sdir> <sname> ? ?

The old-style version of several of the above index file entries allow several file names to be listed in one entry. These "multiple-name" entries are as follows:

cus_sg:	**cus_ss:**	**db:**	**doc:**	**icon:**
ih:	**mi:**	**sg_obj:**	**ss_obj:**	**view:**

When manually converting any of these multiple-name index file entries, replace it with ***several individual*** entries, one per **file:** name in the original entry. Similarly, you must replace **comp:** and **dep:** entries having multiple arguments with several **comp:** or **dep:** entries, each having a single argument.

Automatic Index File Conversion

To automatically convert an index file (<mmname>.i) from old-style format to new-style format, use the following command sequence:

```
mkmmconv <mmname>.i
```

This command will create a new file, <mmname>.i.conv, equivalent to <mmname>.i, but in new-style format. You should then replace the contents of <mmname>.i in your source control system with the contents of <mmname>.i.conv. If your project also involves the generation of a PIB contribution file, you should also add that file under source control.

Index File Conversion Example

The following example illustrates the use of the mkmmconv tool in upgrading old-style index files to SPECTRUM 4.x/5.0 format. (Keep in mind that all

index files must be upgraded to SPECTRUM 4.x/5.0 format, or `mkmm` will refuse to process them.)

Suppose you have a fictitious index file, `DCNET.i`, as delineated in Figure 3-2, which needs to be upgraded to SPECTRUM 4.x/5.0 format in a source development area.

Figure 3-2. Text for DENET.i Index File (Fictitious)

```
  mm: Decnet Management Module
  vend: Cabletron
  rev: 1.01.00
  level: 1
  irev: 01.00.00.000

  iib: CSIib/DecAgent/AgntDnArr.Bsc
  iib: CsIib/DecAgent/DecAgntAL.Bas
  gib: CsGib/DecAgent/CsConfig.30
  gib: CsGib/DecAgent/CsCreate.100
  image: CsImage/DecAgent/decnet_b.csi
  image: CsImage/DecAgent/decnet_g.csi
  pib: CsPib/CsTRDev.pib
CtTokenRingApp/  CtTknApp.DIBM,CtTokenRingApp
  pib: CsPib/CsDevTop.pib TRLbSRPort/LbSRPrt.DIBMh,TRLbSRPort
  pib: CsPib/CsFind.pib CtTokenRingApp/CtTRApp.Bas,CtTokenRingApp
  pib: CsPib/CsDefault.pib
CtTokenRingApp/  CtTRApp.Bas,CtTokenRingApp
  ih: ${DEVAREA}/dcnet/intel/CsIHDecnet.o
  mi: ${DEVAREA}/dcnet/intel/CsDecnetMI.o
  db: ${DEVAREA}/dcnet/db/SM-DEC1001.e
  view: ${LOCAL_NODE}/dcnet1.1/view/CsDecMView.o
  cus_ss: decagent.cus
  file: Install mmdesc ? DCNET.mmd ? ?
  dep: ACME_SMEPI-CSI
```

To convert this index file, you must change to the directory containing `DCNET.i` and then run the `mkmmconv` utility on that index file:

```
   cd
   mkmmconv DCNET
```

(assuming that `mkmmconv` is in your search path).

For the purposes of this example, suppose that `mkmmconv` detects no errors in `DCNET.i`. Therefore, the utility would output the following messages:

```
   Created PIB contribution file DCNET.p
   Created converted index file DCNET.i.conv
```

These messages indicate that the utility has generated the PIB contribution file, DCNET.p, together with a converted index file, DCNET.i.conv.

If mkmmconv detects errors, you would receive one or more error messages, and you would have to fix DCNET.i manually and then rerun mkmmconv.

Figure 3-3 shows the contents of the converted index file, DCNET.i.conv., as it would be produced by the mkmmconv utility for the example inputs.

Figure 3-3. Text for DC NET.i.conv Index File (Fictitious)

```
mm:     Decnet Management Module
vend:   Cabletron
rev:    1.01.00
level:  1
irev:   01.00.00.000

file:   SG iib ? CSIib/DecAgent/AgntDnArr.Bsc ? ?
file:   SG iib ? CsIib/DecAgent/DecAgntAL.Bas ? ?
file:   SG gib ? CsGib/DecAgent/CsConfig.30 ? ?
file:   SG gib ? CsGib/DecAgent/CsCreate.100 ? ?
file:   SG image ? CsImage/DecAgent/decnet_b.csi ? ?
file:   SG image ? CsImage/DecAgent/decnet_g.csi ? ?
file:   SG pib ? DCNET.p ? ?
file:   SS ih ${DEVAREA}/dcnet/intel CsIHDecnet.o ? ?
file:   SS mi ${DEVAREA}/dcnet/intel CsDecnetMI.o ? ?
file:   SS db ${DEVAREA}/dcnet/db SM-DEC1001.e ? ?
file:   SG view ${LOCAL_NODE}/dcnet1.1/view CsDecMView.o ? ?
file:   SS cus ? decagent.cus ? ?
file:   Install mmdesc ? DCNET.mmd ? ?
dep:    ACME_SMEPI-CSI
```

Note that, because there were **pib:** entries in the original DCNET.i file, mkmmconv also created the PIB contribution file, DCNET.p, which contains the information from all the **pib:** entries. Figure 3-4 shows the contents of DCNET.p.

Figure 3-4. Contents of DCNET.p File (Fictitious), as Generated by mkmmconv

```
CsTRDev.pib     CtTokenRingApp/CtTknApp.DIBM,CtTokenRingApp
CsDevTop.pib    TRLbSRPort/LbSRPrt.DIBMh,TRLbSRPort
CsFind.pib      CtTokenRingApp/CtTRApp.Bas,CtTokenRingApp
CsDefault.pib   CtTokenRingApp/CtTRApp.Bas,CtTokenRingApp
```

Note that the **pib:** entries do not appear in the converted index file, DCNET.i.conv. Instead, that file has a "**file:** SG pib ? DCNET.p ? ?" entry, which is used to ship the DCNET.p file, which now contains the **pib:** entry information.

While mkmmconv guarantees that every line of DCNET.i.conv is in SPECTRUM 4.x/5.0 index file format, it does not guarantee that DCNET.i.conv as a whole is acceptable to mkmm. For example, required entries may be missing, certain entries that should be unique may be duplicated, etc. Therefore, you should check DCNET.i.conv by using mkmm to generate an extension module archive from it.

First, visually compare the contents of DCNET.i.conv and DCNET.p with the contents of the original DCNET.i until you are satisfied that the conversion has been performed properly. Then replace the original index file DCNET.i (after saving that replaced file under a temporary file name for safety sake) with the converted index file DCNET.i.conv as follows:

```
mv DCNET.i. DCNET.i.old
mv DCNET.i.conv DCNET.i
```

Then, invoke mkmm to generate the extension module archive DCNET.tar.

```
mkmm DCNET.i
```

If mkmm generates any errors (see the *mkmm and mkmmdeps Errors* discussion on page A-6), address these errors by manually modifying DCNET.i, and then rerunning mkmm as described above until no more errors are generated.

Once you are confident that the new version of DCNET.i is correct, delete the saved copy of the original version (DCNET.i.old) and then complete the upgrade of DCNET.i to the SPECTRUM 4.x/5.0 format as follows:

- Replace the original contents of the index file DCNET.i in your source control area with the converted contents.
- Add the PIB contribution file, DCNET.p, to your source control area. (If DCNET.i did not have any **pib:** entries, mkmmconv would not have generated DCNET.p, and this conversion process would not have been necessary.)

Chapter 4

Tools

This chapter describes the tools provided by the SEI Toolkit and explains their requirements, options, and usage, as well as indicating "retired" features. The discussion also includes examples for creating various types of SPECTRUM media.

Special Requirements

This section describes some special requirements for using the tools described in this chapter. Please read this section carefully before using any of these tools.

SEI Toolkit Version Compatibility

As of Version 4.0, the SEI Toolkit is not compatible with earlier versions of the SEI Toolkit. For more detailed information, please read the precautionary information in *SEI Toolkit Versioning*, starting on page 1-1.

SEI Toolkit and Windows NT

When using the SEI Toolkit tools on the Windows NT platform, some special considerations apply:

- On the Windows NT platform, the proper behavior of tools in the SEI Toolkit requires prior installation of NuTCracker UNIX support software. Normally, this will not be a problem, as the required NuTCracker software is shipped on each SPECTRUM CD-ROM and is installed prior to installing SPECTRUM. However, interim removal or modification of the installed NuTCracker software can result in failure of SEI Toolkit tools (as well as other SPECTRUM software).

- On the Windows NT platform, it is necessary to run the NuTCracker Korn Shell (ksh) before invoking any of the tools in the SEI Toolkit tools. *This must be done even when invoking SEI Toolkit tools from within a NuTCracker Korn Shell window.* You must also keep this requirement in mind when invoking SEI Toolkit tools from Makefiles or other build management tools.

SEI Toolkit Tools

The following sections describe the tools used during each phase of a product's evolution, explaining where each such tool can be found.

Tools in the SEI Toolkit are classified into three groups, according to the phase of product evolution in which they are used. The tools used in each phase are:

Development: mkmmconv, mkmmdeps, mkmm

Manufacturing: mkmanf, mkarea, mkdist

Integration: Install, Install.quick

For an overview of the product architecture and evolution phases, see Chapter 2, *Toolkit Architecture*.

Development Tools Specification

This section describes the tools used in the development phase.

Development: mkmmconv

Usage: **mkmmconv [picky=y/n] mm=<mmname> \ [<mmname> …]**

Description: As of SPECTRUM 4.0, the format of extension module index files changed such that index files compatible with older versions of the SEI Toolkit may no longer be compatible with the SPECTRUM 4.x/5.0 SEI Toolkit, specifically with mkmm. Consequently, you must convert pre-existing index files to the new format. You can use the mkmmconv tool to perform most of this conversion, preserving all relevant information in the original index file, including comments. The primary use of mkmmconv is to upgrade existing index files for compatibility with SPECTRUM 4.x/5.0.

Using mkmmconv *to convert an index file guarantees only that each entry of the index file is syntactically correct for SPECTRUM 5.0 and does not guarantee that the file as a whole will be acceptable to the SPECTRUM 5.0 versions of* mkmmdeps *or* mkmm*. To be completely sure that a converted index file is correct, you must use* mkmm *to build an extension module archive from the index file, addressing any errors generated by* mkmm*, as discussed in* *Index File Conversion, starting on page 3-37**.*

Running the mkmmconv tool has the same functionality as using the **mkmm convert=y** command in previous versions of the SEI Toolkit.

System Environment: The mkmmconv utility must reside in its original location, as determined when the SEI Toolkit is installed, but must be invoked from the directory containing the index file(s) that are to be converted.

Command Line Options: The following options are available for the mkmmconv tool; note that some can be used in combination while some are mutually exclusive.

picky=n (Default) Report only definite errors.

picky=y In addition to reporting definite errors found in the index file during conversion, warn about other probable errors.

mm=<mmname> Convert index file <mmname>.i. (The ".i" extension may be included in <mmname>.) Several index files may be converted at the same time by including several **mm=<mmname>** options on the same ***mkmmconv*** command line.

<mmname> Equivalent to **mm=<mmname>**. Matches any command line argument not recognized as any other option.

In the absence of both **mm=<mmname>** or **<mmname>** options, mkmmconv does nothing; one or the other must be present.

Input File: mkmmconv uses the following input files:

index files For each extension module <mmname> specified on the command line, mkmmconv attempts to convert the index file <mmname>.i.

Output Files: mkmmconv may generate the following output files:

converted index files For each extension module <mmname> specified on the command line, mkmmconv attempts to convert the index file, <mmname>.i, to SPECTRUM 5.0 index file format. If successful, this operation produces the converted index file, <mmname>.i.conv.

When upgrading a development source for compatibility with SPECTRUM 5.0, you should replace the contents of <mmname>.i *with the contents of* <mmname>.i.conv *in your source control area. Otherwise, the index file will not be compatible with SPECTRUM 5.0 SEI Toolkit tools.*

PIB contribution files: When mkmmconv converts an index file (<mmname>.i) that contains **pib:** entries, the relevant information from the **pib:** entries is placed in a PIB contribution file, created as <mmname>.p by the operation of mkmmconv. In the converted index file, the **pib:** entries are replaced with a single **file: SG pib** entry, which ships the PIB contribution file. More on PIB contribution files may be found in *PIB Contribution Files*, starting on page 5-30.

When upgrading your development source area for compatibility with SPECTRUM 5.0, you should add the file <mmname>.p to the source control area if you have already done that for compatibility with SPECTRUM 4.0. Otherwise, the mkmm tool will fail when processing the "**file: SG pib**" entry in the converted ind*ex file*.

Error and warning files: Any errors generated while mkmmconv is converting <mmname>.i are saved in a file created as <mmname>.err. Similarly, mkmmconv saves any warnings generated during that process in a file created as <mmname>.warn.

Exit Status: If mkmmconv is unable to convert all index files specified on its command line, it returns with exit status 0, otherwise it returns with exit status 1.

Errors: mkmmconv errors are described in *mkmmconv Errors*, starting on page A-2.

Development: mkmmdeps

Usage: **`mkmmdeps [picky=y/n] [exts=<extlist>] \ mm=<mmname> [<mmname> …]`**

Description: Some extension module developers manage their extension module build process with the UNIX `make` tool. Such developers may wish to use Makefiles to manage the generation of extension module archives via the `mkmm` tool. To do this properly, it is first necessary to generate the Makefile dependencies of the extension module archives with respect to their index files and shippable files.

The `mkmmdeps` tool prints a list of Makefile dependencies for one or more extension module archives in Makefile format. You can then embed this dependency list manually or dynamically into a Makefile. Since the process of integrating the dependency list into a Makefile varies across development sites, it will not be discussed in this document

Do not invoke `mkmmdeps` *to generate Makefile dependencies for an extension module archive until after all shippable files for that extension module have been built. Otherwise, the dependencies may not be correct. This problem is especially likely on Windows NT. For further explanation, see the section entitled* *Source File Extension Handling in mkmm and mkmmdeps, starting on page 4-13.*

System Environment: The `mkmmdeps` utility must reside in its original installed location within the SEI Toolkit, but must be invoked in the directory containing the index file(s) for which the Makefile dependencies are to be generated.

Environment Variables: Any environment variable included explicitly in an index file must be exported before invoking `mkmmdeps`. Also, if an index file uses a default value that references an environment variable, the referenced environment variable must be exported before invoking `mkmmdeps`.

Command Line Options: The following options are available for the `mkmmdeps` tool; note that some can be used in combination while others are mutually exclusive.

`picky=y` In addition to reporting definite errors found in the index file, warn about other probable errors.

`picky=n` (Default) Report only definite index file errors.

exts=<extlist>
Use the <extlist> list of filename extensions when looking for shippable files. By default, <extlist> is "**,.exe,.cmd,.bat**." Using filename extension lists helps make index files more portable between Solaris and NT platforms. For more information on this option and filename extension lists, refer to *Source File Extension Handling in mkmm and mkmmdeps*, starting on page 4-13.

mm=<mmname> Generate dependencies for the extension module archive, <mmname>.tar. **<mmname>** may be the name of an index file, in which case you would not use the ".i" extension. You can generate dependencies for several extension module archives by including several **mm=<mmname>** options on the same mkmmdeps command line.

<mmname> Equivalent to **mm=<mmname>**. Matches any command line argument not recognized as any other option.

In the absence of both mm=<mmname> or <mmname> options, mkmmdeps does nothing; one or the other must be present in order for any effect to result.

Input Files: mkmmdeps uses the following input files:

index files For each extension module, <mmname>, specified on the command line, mkmmdeps reads the index file, <mmname>.i.

shippable files The mkmmdeps tool checks for the existence of each shippable file mentioned in the **file:** and **head:** entries of each index file it processes.

mkmm.sys and ***mkmm.loc*** The mkmmdeps tool reads the mkmm.sys and mkmm.loc data files to determine the set of file types that are legal in SPECTRUM extension module index files, along with the default characteristics (including source and target directories, etc.) associated with

those file types. For more information on how `mkmmdeps` uses `mkmm.sys` and `mkmm.loc`, see the section entitled *mkmm.sys and mkmm.loc*, starting on page 4-11.

Output Files: `mkmmdeps` prints the Makefile dependency list to `stdout`. It also generates the following output files:

error and ***warning files***	Any errors or warnings generated by `mkmmdeps` while generating Makefile dependencies for extension module archive `<mmname>.tar` are saved in the files `<mmname>.err` and `<mmname>.warn`, respectively.

Exit Status: If `mkmmdeps` is able to generate dependencies for all index files specified on its command line, it returns with exit status 0, otherwise it returns with exit status 1.

Errors: `mkmmdeps` errors are described in Appendix A, in *mkmm and mkmmdeps Errors*, starting on page A-6.

Development: mkmm

Usage:

```
mkmm [picky=y/n] [exts=<extlist>] \
mm=<mmname> [<mmname> …]
```

Description: The `mkmm` tool uses an extension module index file to build a SPECTRUM extension module archive file containing the shippable files associated with the extension module. You can add the resulting extension module archive file to a manufacturing area for distribution on a SPECTRUM medium (refer to the discussions of `mkarea` on page 4-20 and of `mkdist` on page 4-23). You can also install the resulting file directly on top of an existing SPECTRUM area (see the `Install.quick` discussion on page 4-32). You can use a single invocation of `mkmm` to generate several extension module archives from several index files. For more information on index files, see Chapter 3, *Index Files*.

Extension module archives produced by `mkmm` are in `tar` archive format. The name of the extension module archive depends on the name of the index file used to produce it. If the index file is named `SMACME.i`, for example, `mkmm` will produce an extension module archive named `SMACME.tar`.

Do not rename any extension module archive produced by `mkmm`*. The* `mkarea` *and* `Install.quick` *programs will process renamed extension module archives improperly, which can result in a corrupted manufacturing area.*

System Environment: The `mkmm` utility must reside in its originally installed location within the SEI Toolkit, but it must be invoked in the directory containing the index file(s) from which the extension module archive file(s) are to be built.

Environment Variables: Any environment variable included explicitly in an index file must be exported before invoking `mkmm`. Also, if an index file uses a default value that references an environment variable, you must export the referenced environment variable before invoking `mkmm`.

Command Line Options:

The following options are available for the `mkmm` tool; note that some can be used in combination, while others are mutually exclusive.

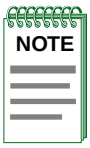
`picky=n` (Default) Report only definite index file errors.

`picky=y` In addition to reporting definite errors found in the index file, warn about other probable errors.

`exts=<extlist>`
Use the `<extlist>` list of filename extensions when looking for shippable files. By default, `<extlist>` is "**`,.exe,.cmd,.bat`**." Using filename extension lists helps make index files more portable between Solaris and NT platforms. For more information on this option and filename extension lists, see the section entitled *Source File Extension Handling in mkmm and mkmmdeps*, starting on page 4-13.

`mm=<mmname>` Generate an extension module archive, to be named `<mmname>.tar`. `<mmname>` may be the name of an index file, in which case you would not use the "`.i`" extension. You can generate several extension module archives at once by including several **`mm=<mmname>`** options on the `mkmm` command line.

<mmname> Equivalent to mm=<mmname>. Matches any command line argument not recognized as any other option.



If neither **mm=<mmname>** nor **<mmname>** options are present, mkmm does nothing; one or the other must be present in order for any results to occur.

Input Files: ***mkmm*** uses the following input files:

index files For each extension module, <mmname>, specified on the command line, mkmm reads the index file, <mmname>.i.

shippable files The mkmm tool reads each shippable file mentioned in the **file:** and **head:** entries of each index file it processes.

mkmm.sys and ***mkmm.loc*** The mkmm tool reads the mkmm.sys and mkmm.loc data files to determine the set of file types legal in SPECTRUM extension module index files. The tool also reads the default characteristics (including source and target directories, etc.) associated with those file types. For more information on how mkmm uses mkmm.sys and mkmm.loc, see the section entitled *mkmm.sys and mkmm.loc*, starting on page 4-11.

Output Files: The ***mkmm*** tool creates the following files:

extension module archives For each extension module (**<mmname>**) that you specify on the command line, the mkmm tool creates an extension module archive, named <mmname>.tar. Each such *.tar archive contains all files necessary to successfully ship and install the **<mmname>** extension module, including all shippable files specified in the index file, <mmname>.i.

error and ***warning files*** Any errors or warnings generated by mkmm while generating the extension module archive <mmname>.tar are saved in the <mmname>.err and <mmname>.warn files, respectively.

Exit Status: If mkmm is able to generate extension module archives for all index files specified on its command line, it returns with exit status 0, otherwise it returns with exit status 1.

Errors: mkmm errors are described in Appendix A, in *mkmm and mkmmdeps Errors*, starting on page A-6.

Retired Features:

The following ***mkmm*** features, prominent in earlier versions of the SEI Toolkit, were retired as of SPECTRUM 4.0:

check, check=y, check=n, $MM_VERSION
The **check, check=y,** and **check=n** command line options were formerly used to enable and disable checksum protection of SpectroGRAPH GIB and IIB files. When using the **check** or **check=y** options, it was necessary to set and export the environment variable, $MM_VERSION, beforehand. As of SPECTRUM 4.0, a new checksumming mechanism was implemented. It is no longer necessary to set and export $MM_VERSION, and these options have been retired.

convert=y, convert=n
The **convert=y** and **convert=n** command line options were formerly used to turn on or to turn off conversion of index files to newer formats. Conversion of index files to the formats used in SPECTRUM 5.0 is accomplished by the mkmmconv tool. Use of mkmm requires that any pre-4.0 index files be converted to SPECTRUM 4.x/5.0 format.

depend=y, depend=n
The **depend=y** and **depend=n** command line options were formerly used to turn generation of extension module archive Makefile dependencies on or off. Generation of Makefile dependencies for extension module archives is now accomplished by the mkmmdeps tool.

$ext, $vend mkmm formerly used two environment variables, $ext and $vend, as the standard default source directories for several shippable file types. In SPECTRUM 5.0, as was true with SPECTRUM 4.0, the standard default source directory for all shippable file types is "." (the directory in which mkmm is invoked). If your extension module development environment depends heavily on the old behavior of mkmm with regard to $ext and $vend, you may restore it by adding the following lines to the file mkmm.loc, located in the same directory as mkmm (if mkmm.loc does not exist, create it):

```
deflt: SG gib sdir ${ext}
deflt: SG iib sdir ${ext}
deflt: SG image sdir ${ext}
deflt: SG other sdir ${ext}
deflt: SG rib sdir ${ext}
deflt: SS vfile sdir ${vend}
```

The braces used with `${ext}` in this example are not actually necessary (refer to the *Environment Variables* discussion on page 3-2), but developers would be wise to adopt this habitual caution so as to ensure correct expansion of variables in the index files.

Alternatively, you can use **`deflt:`** entries in the index files to override the inappropriate default source directories.

`mkmm.sys` and `mkmm.loc`

The `mkmm` and `mkmmdeps` tools use two data files, `mkmm.sys` and `mkmm.loc`, to define the set of file types that may legally be used in **`file:`** and **`head:`** entries in index files, as well as the standard default characteristics (including source directory, target directory, etc.) associated with each file type. Through proper use of these files, extension module developers can customize the default behavior of `mkmm` or `mkmmdeps` when processing **`file:`** and **`head:`** entries. Specifically, extension module developers can change the standard default source directories associated with any given file type.

When processing an index file, `mkmm` and `mkmmdeps` both go through the following steps:

1. Process `mkmm.sys`.

   The `mkmm.sys` file is in index file format and contains **`deflt:`** entries for each file type that may legally be used in an index file. It should contain no other entries. These **`deflt:`** entries establish the standard default values (source directory, target directory, and checksum protection status) associated with each file type that may legally be used in an index file.

2. Process `mkmm.loc` (if present).

   The `mkmm.loc` file also is in index file format and contains only **`deflt:`** entries. Since `mkmm` and `mkmmdeps` process the `mkmm.loc` entries after the `mkmm.sys` entries, any same-application **`deflt:`** entries in `mkmm.loc` override the standard default values set by the **`deflt:`** entries in `mkmm.sys`. This arrangement thus lets the developer customize standard default values associated with file types, by adding the desired override commands to the `mkmm.loc` file.

3. Process the index file.

The standard default values for the various file types defined in `mkmm.sys` and `mkmm.loc` now apply to the index file. The **`deflt:`** entries in the index file may be used to override these default values, however. The **`deflt:`** entries in `mkmm.sys` and `mkmm.loc` apply to all index files processed by `mkmm` or `mkmmdeps`, whereas **`deflt:`** entries in a given index file apply only to that index file.

Cabletron ships the `mkmm.sys` file with the SEI Toolkit, and the extension module developer should never remove or modify this file. Removal or improper modification of `mkmm.sys` can cause `mkmm` or `mkmmdeps` failure and can result in production of extension modules that cannot be installed. Moreover, future upgrading of the SEI Toolkit would overwrite `mkmm.sys`, undoing any interim modifications.

Conversely, Cabletron does not ship `mkmm.loc` with the SEI Toolkit. Instead, the extension module developer can create `mkmm.loc` (either directly or initially as a copy of `mkmm.sys`) and edit it manually to customize the standard default values in `mkmm` and `mkmmdeps`. Upgrades of the SEI Toolkit will not affect the contents of `mkmm.loc`.

Both `mkmm.sys` and `mkmm.loc` should reside in the same directory in which `mkmm` and `mkmmdeps` tools reside.

The following example illustrates the use of `mkmm.loc` to customize standard default values processed by `mkmm` and `mkmmdeps`.

As shipped, the `mkmm.sys` file contains the following **`deflt:`** entries:

```
deflt: SG stdmenu sdir .
deflt: SG stdmenu tdir SG-Tools/stdmenu req=yes
```

These entries say that:

- Files of product "`SG`" and file type "`stdmenu`" (that is, "`SG stdmenu`" files, pertaining to SpectroGRAPH standard menu setups) are legal in SPECTRUM extension modules.
- The standard default source directory (`sdir`) of "SG stdmenu" files is "`.`" — meaning that `mkmm` and `mkmmdeps`, by default, look for "`SG stdmenu`" files in the directory "." (that is, in the same directory in which `mkmm` or `mkmmdeps` was invoked).
- The standard default target directory (`tdir`) is "`SG-Tools/stdmenu`" — meaning that `mkmm`, by default, packages "`SG stdmenu`" files in the extension module archive, so that `Install` subsequently can extract them from the medium into the SG-Tools/stdmenu directory (relative to the `Install` root directory).
- The standard default target directory (`tdir`) of "`SG stdmenu`" files cannot be overridden (`req=yes`) by later "**`deflt:`**" entries in `mkmm.loc` or the index file.

Now suppose that the source for "SG stdmenu" files at a particular development site were kept in the `/user/devarea/stdmenu_files` directory. At that site, it would be convenient to arrange for `mkmm` and `mkmmdeps` to look for "`SG stdmenu`" files in that location. To get this to happen, the developer should add the following entry to `mkmm.loc`:

```
deflt: SG stdmenu sdir /user/devarea/stdmenu_files
```

This entry then will override the "**`deflt: SG stdmenu sdir .`**" entry in `mkmm.sys`, redefining the standard default source directory for "`SG stdmenu`" files to `/user/devarea/stdmenu_files`. This new default identification will now apply whenever `mkmm` or `mkmmdeps` is used to process any index file.

For more detailed information on **`deflt:`** entries, refer to the **`deflt:`** label specifications, beginning on page 3-16.

Source File Extension Handling in mkmm and mkmmdeps

This section applies primarily to developers of extension modules for the Windows NT platform.

There are several file naming conventions that differ between UNIX and Windows NT file systems. The most notable is that certain types of files on NT, such as executable files and batch files, require specific filename extensions (for example — `.bat`, `.com`, `.cmd`, `.exe`, `.ksh`, etc.), whereas this typically is not the case on UNIX systems.

These differences can cause inconveniences in the index files for multi-platform extension modules that ship on both UNIX and NT platforms. For instance, a compiled executable called `mytool` on UNIX would be called `mytool.exe` on NT. In an extension module index file prepared for shipping such an executable, therefore, there would apparently need to be two separate **`file:`** entries, one for UNIX platforms and one for NT, as follows:

```
file: SS obj ? mytool ? ? plat!=intel
file: SS obj ? mytool.exe ? ? plat=intel
```

This solution is unsatisfactory, because it potentially requires a lot of duplicated entries in the index files.

An alternative solution would be to use a `${EXE}` environment variable, set to the null string on UNIX and to "`.exe`" on NT. Given this setup, the single index file entry:

```
file: SS obj ? mytool${EXE} ? ? plat=intel
```

would suffice to ship the file on both Solaris and NT platforms. Deciding exactly which files need an extension would still be a chore when many files were involved, however.

To alleviate this burden for the extension module developer, the `mkmm` and `mkmmdeps` tools automatically check for standard file name extensions when processing **`file:`** and **`head:`** entries.

For example, suppose we were using `mkmm` to generate an extension module archive. Suppose further that the extension module index file contained the following entry:

```
file: SS obj ? mytool ? ?
```

When `mkmm` processed this entry, it would look for the source file `mytool` with a number of standard extensions. The default list of extensions is:

No extension (null)
`.exe`
`.cmd`
`.bat`

Accordingly, `mkmm` would look first for source file `mytool`, then `mytool.exe`, then `mytool.cmd`, and finally `mytool.bat`. On a Solaris platform, then, it would find and ship `mytool`, whereas on an NT platform, where `mytool` does not exist, `mkmm` would find and ship `mytool.exe`. In this typical instance, therefore, the single **`file:`** entry works without change on either UNIX or NT.

In the rare case that it may be necessary, the list of extensions used by `mkmm` or `mkmmdeps` when processing **`file:`** or **`head:`** entries can be changed by using the **`exts=<extlist>`** option, where `<extlist>` is a comma-separated list of extensions to be checked. For instance, the command:

```
mkmm exts=,.exe,.com DCNET.i
```

instructs `mkmm` to check first for no extension (because there is nothing in `<extlist>` before the first comma), then for an `.exe` extension, and finally for a `.com` extension when processing **`file:`** and **`head:`** entries in index file `DCNET.i`.

If either `mkmm` or `mkmmdeps` exhausts the extension list without finding the source file, it will use the source file name with no extension. For `mkmm`, this means it will not find the source file and therefore will fail; this result is appropriate for `mkmm`, because this sequence means that the source file is in fact missing.

This situation will not bother `mkmmdeps`, however, and the unextended source file will end up in the Makefile dependency list for the extension module

archive. This situation can be a problem, as demonstrated in the following scenario.

Suppose you have an extension module development area for the Windows NT platform, in which you build the executable `mytool.exe` for shipment in your extension module. Suppose further that, during your build, you use `mkmmdeps` to generate a Makefile dependency list for your extension module archive. If you invoke `mkmmdeps` before `mytool.exe` is built, `mkmmdeps` will fail to find `mytool.exe` and will therefore put the unextended file name `mytool` into the Makefile dependency list rather than `mytool.exe`. Even if `mytool.exe` is properly built later, the extension module archive will fail to build, because it depends on the nonexistent `mytool` file. For this reason, `mkmmdeps` should not be invoked until all shippable files in the extension module have been built — and preferably should be built just before the extension module archive is built. This is especially necessary on the Windows NT platform.

Manufacturing Tools Specification

This section describes the tools that merge a SPECTRUM extension module into the manufacturing area from the development environment. These tools define the manufacturing area directory structure and put the necessary pieces, as defined in the index file, in their designated places. This area then puts the SPECTRUM extension module onto the release media.

Manufacturing: `mkmanf`

Usage: **`mkmanf [compress=gzip] [uncompress=yes]`**

Description: The main purpose of the `mkmanf` tool is to create a manufacturing area. A manufacturing area is a repository for SPECTRUM software from which SPECTRUM distribution media (virtual tapes) may be generated for later installation. For more information on manufacturing areas and their use in the manufacturing process, refer to the *Manufacturing Phase* discussions on page 2-4.

Manufacturing areas exist in two forms, uncompressed and compressed. In an uncompressed manufacturing area, extension module records are uncompressed; in a compressed manufacturing area, they are compressed. A compressed manufacturing area takes up less disk space, and produces smaller distribution media (virtual tapes). In turn, these media space savings result in faster media installation. For large distributions, the space and time savings are especially significant.

`mkmanf` can be used to convert an existing uncompressed manufacturing area into a compressed manufacturing area, and *vice versa*.

System Environment: The `mkmanf` utility must reside in its originally installed location within the SEI Toolkit.

Environment Variables: The following environment variables affect the behavior of `mkmanf`:

`$MANF` Prior to invoking `mkmanf`, the environment variable `$MANF` must be set and exported. `$MANF` must be set to the absolute path name of a writable directory in which a manufacturing area is to be created or already exists.If this is not done, `mkmanf` will fail.

If a manufacturing area does not exist in the `$MANF` directory, `mkmanf` creates one there. If a manufacturing area already exists in the `$MANF` directory, `mkmanf` converts it to a compressed or uncompressed manufacturing area, as specified by its command line options.

Command Line Options: The following options are available for the `mkmanf` tool:

`compress=gzip` Create a compressed manufacturing area, or convert an existing uncompressed manufacturing area to a compressed manufacturing area, according to the value of the environment variable `$MANF` *(which see, above)*. During conversion, extension module records in the manufacturing area are compressed using the GNU `gzip` data compression utility.

`uncompress=y` Create an uncompressed manufacturing area, or convert an existing compressed manufacturing area to an uncompressed manufacturing area, according to the value of the environment variable `$MANF` *(which see, above)*. During conversion, extension module records in the manufacturing area are uncompressed by the use of the GNU `gzip` data compression utility.

The **compress=gzip** and **uncompress=y** options should not be used together on the mkmanf command line. If mkmanf is used to create a manufacturing area, and neither **compress=gzip** nor **uncompress=y** options are specified, the created manufacturing area is uncompressed. Manufacturing area compression by utilities other than gzip is not supported.

Note that when the mkarea tool is subsequently used to add extension module archives to a compressed manufacturing area, the extension modules records are automatically compressed as they are added.

restore=y Upgrade the Install support software in an existing manufacturing area within the $MANF directory. When a new minor version of the SEI Toolkit is installed for use at a VAR development site, mkmanf should be invoked with this option to upgrade the Install support software in each manufacturing area.

(When a new major version of the SEI Toolkit is installed, all extension module archives and manufacturing areas should be completely rebuilt from source files, as the existing extension module archives and manufacturing areas will not be compatible with the new major version of Install.)

Input Files: The mkmanf tool uses no input files.

Output Files: The mkmanf tool creates the following output files:

Manufacturing Area Files When invoked in order to create a new manufacturing area, the mkmanf tool creates files and directories within the $MANF directory. Most of these files are support files for the Install utility, which must be present on every SPECTRUM distribution medium generated from the manufacturing area.

Exit Status: If mkmanf is able to create the manufacturing area, it returns with exit status 0; if the attempt is unsuccessful for any reason, mkmanf returns with exit status 1.

Updating the MMML File

When ordering SPECTRUM software from Cabletron, a customer does not buy individual extension modules by their extension module names. Instead, a customer buys groups of extension modules by part numbers. Each such group is called a *purchasable part*. The extension modules in a given purchasable part are installed together, collectively supporting a well-defined subset of SPECTRUM functionality.

The component selection screen of the `Install` GUI allows the installing user to select SPECTRUM software for installation by purchasable part name. During installation, the extension modules in each purchasable part selected during this process are installed together as a group.

The way that extension modules are grouped into purchasable parts is specified by a MMML (**M**aster **M**anagement **M**odule **L**ist) file. The MMML file is created in each new manufacturing area by the `mkmanf` tool.

Unfortunately, the MMML file created by `mkmanf` in the manufacturing area is tailored to the Cabletron development environment, with the result that the file is not appropriate for distributing VAR-developed extension modules (there are plans to fix this problem in future releases). Therefore, it is strongly recommended that VAR developers manually update the MMML file, sometime after running `mkmanf` to create the manufacturing area — but before running `mkdist` to create a SPECTRUM distribution medium from the manufacturing area.

Manually updating the MMML file lets you specify the purchasable part groupings of your extension modules. It will also prevent `mkdist` from generating several pages worth of warning messages similar to:

```
Errors detected in manufacturing area:
Warning: MMML PP SA-CSI1000 MM CORE not in Manf area.
Warning: Removing MM CORE from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM Ibm6611 not in Manf area.
Warning: Removing MM Ibm6611 from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM NODELS not in Manf area.
Warning: Removing MM NODELS from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM Notifier not in Manf area.
Warning: Removing MM Notifier from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM OBackup not in Manf area.
Warning: Removing MM OBackup from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM SG-Support not in Manf area.
Warning: Removing MM SG-Support from MMML PP SA-CSI1000.
...
```

(These error messages also appear when using the `Install.quick` tool, but in that case they do not indicate a problem that needs to be fixed.)

Example for Updating the MMML File

Suppose that you have developed three extension modules called EXTMODA, EXTMODB, and EXTMODC, and that you have already done the following:

- Used `mkmm` to produce the extension module archives `EXTMODA.tar`, `EXTMODB.tar`, and `EXTMODC.tar`.
- Used `mkmanf` to create a manufacturing area.
- Used `mkarea` to add `EXTMODA.tar`, `EXTMODB.tar`, and `EXTMODC.tar` to the manufacturing area.

Suppose further that you now wish to create a distribution medium containing the extension modules EXTMODA, EXTMODB, and EXTMODC, and also that you want EXTMODA and EXTMODB to be always installed together as a unit, and EXTMODC to be separately installable by itself. To achieve this, you must update the MMML file. This is done as follows:

Using any desired text editor, you must edit the MMML file, which you will find identified in the system as `$MANF/<plat>.dist/mmml` (where `$MANF` is the manufacturing area root directory, as described in *Manufacturing: mkmanf*, on page 4-15, and `<plat>` is the name of the relevant platform, as described in *Index File Platforms*, on page 3-5). Replace the existing contents of the MMML file with the following:

```
EMGROUP1: EXTMODA EXTMODB
EMGROUP2: EXTMODC
```

This MMML file identifies (and groups) the extension modules EXTMODA and EXTMODB as the purchasable part, EMGROUP1, and identifies EXMODC as the purchasable part, EMGROUP2.

In the MMML, avoid using purchasable part names that start with "SA-" "SM-" or "ST–" prefixes. Cabletron reserves these purchasable part names for use in its official distributions.

After modifying the MMML file, use `mkdist` to create a SPECTRUM distribution medium containing the extension modules EXTMODA, EXTMODB, and EXTMODC.

Finally, use `Install` to install the medium. The component selection screen of `Install` will allow you to select the purchasable parts EMGROUP1 and EMGROUP2. Selecting EMGROUP1 will result in extension modules EXTMODA and EXTMODB being installed together, and selecting EMGROUP2 will result in the extension module EXTMODC being installed by itself.

Retired Features:

The following `mkmanf` features, prominent in earlier versions of the SEI Toolkit, were retired as of SPECTRUM 4.0:

`compress, compress=, compress gzip`
These command line arguments were formerly all equivalent to **`compress=gzip`**.

`compress=compress, compress compress`
These command line arguments were formerly used to specify compression of extension module records in a manufacturing area using the UNIX `compress` compression utility. Compression of manufacturing areas via `compress` is no longer supported.

Manufacturing: `mkarea`

Usage: **`mkarea [add/extract] [plist=<file>/parts] \ [tar=<dir>] [tmp=<dir>] mm=<mmname> \ [<mmname> …]`**

Description: The `mkarea` tool is most often used to add one or more extension module archives to an existing manufacturing area. It may also be used to extract one or more extension module archives from an existing manufacturing area.

As each extension module archive is added to a manufacturing area, its format is changed in order to facilitate creation of SPECTRUM distribution media from the manufacturing area. Therefore, once an extension module archive has been added to a manufacturing area, it is a nontrivial task to remove and rebuild the extension module archive, which is why the **`extract`** option is supplied.

If the manufacturing area is compressed, the software in the extension module archives will be compressed as it is added to the manufacturing area. This will reduce the size of SPECTRUM distribution media created from the manufacturing area.

System Environment: The `mkarea` tool must reside in its originally installed location within the SEI Toolkit. When `mkarea` is invoked to add/extract extension module archives to/from an existing manufacturing area, two conditions must exist. First, the current directory must be the root directory of the manufacturing area (as specified via the `$MANF` environment variable when `mkmanf` was used to create the manufacturing area). Second, all of the extension module

archives to be added to the manufacturing area from the current directory or to be extracted from the manufacturing area must already exist in the applicable source directory.

Command Line Options:

The following options are available for the `mkarea` tool:

`add` (Default) Add extension module archives to the manufacturing area from the current directory. After each specified extension module archive is added to a manufacturing area, it is deleted from the current directory.

`extract` Extract extension module archives from the manufacturing area. Each specified extension module is removed from the manufacturing area and packaged into an extension module archive.

The **`add`** and **`extract`** options should not be used together on the `mkarea` command line. If the **`extract`** option is specified, the **`tmp=<dir>`** and **`tar=<dir>`** options must also be specified.

`plist=<file>`
Generate parts list `<file>`. If you specify this option, `mkarea` puts the names of all extension modules in the manufacturing area into `<file>`. If unspecified, no parts list is generated. `<file>` may be an absolute or relative path. If `<file>` is relative, it is taken as being relative to the manufacturing area root directory. If `<file>` already exists, it is overwritten.

`parts` Same as **`plist=parts`**. Generates a parts list file called `parts` in the manufacturing area root directory.

Only one **`plist=<file>`** or **`parts`** option should appear on the `mkarea` command line.

`mm=<mmname>` Add/extract the extension module archive `<mmname>.tar` to/from the manufacturing area. `<mmname>` may end with the `.tar` extension.

`<mmname>` Same as **`mm=<mmname>`**. Matches any argument not recognized as another option.

Several **mm=<mmname>** and **<mmname>** options may be given on the mkarea command line. This will cause mkarea to add/extract each specified extension module archive to/from the manufacturing area, according to whether the **add** or **extract** option is in effect.

If no **mm=<mmname>** or **<mmname>** option is given, and the **add** option is in effect, mkarea will add all extension module archives existing in the manufacturing area root directory to the manufacturing area. If the **extract** option is in effect, mkarea will do nothing.

tar=<dir> When extracting extension module archives from a manufacturing area, create any extension module archives in the directory <dir>. (The directory <dir> must already exist and must be writable.)

tmp=<dir> When extracting extension module archives from a manufacturing area, create any temporary files within directory <dir>. (The directory <dir> must already exist and must be writable.)

Input Files: The mkarea tool uses the following input files:

Extension module archives
When the **add** option is in effect, mkarea reads and deletes extension module archives in the manufacturing area root directory.

Manufacturing area files
When the **extract** option is in effect, mkarea reads and removes files associated with extension modules from the manufacturing area.

Output Files: The mkarea tool generates the following output files:

Extension module archives
When the **extract** option is in effect, ***mkarea*** creates extension module archives in the manufacturing area root directory.

Manufacturing area files
When the **add** option is in effect, mkarea adds files associated with specified extension modules to the manufacturing

area.

Exit Status: If `mkarea` is able to add all extension module archives specified on its command line to the manufacturing area, it returns with exit status 0. If this task cannot be successfully completed for any reason, `mkarea` returns with exit status `1`.

Manufacturing: `mkdist`

Usage: **`mkdist [clobber=y/n] [comp=y/n] [dep=y/n] [level=0/1/2] [plist=<file>] [prod=SS/SG] \ [remmm=y/n] [v`**tape**`=<dir>]`**

Description: The `mkdist` tool creates a SPECTRUM distribution by copying SPECTRUM extension module software from a manufacturing area to a SPECTRUM distribution medium. SPECTRUM distributions are placed on virtual tape, also called *vtape* (a directory structure on disk containing files organized like a physical SPECTRUM medium, which can be installed in the same manner as a physical tape). For more information about creating a vtape, refer to *Vtape Creation Example*, on page 4-29. Once an extension module has been committed to a distribution medium, it can be installed by the use of the `Install` installation tool.

The `mkdist` *utility attempts to modify and create files in the manufacturing area. If you try to distribute SPECTRUM software to a medium from a read-only manufacturing area, therefore,* `mkdist` *will fail. To avoid this situation, make sure the manufacturing area from which you are distributing SPECTRUM software is not set for read-only operation.*

System Environment:

The `mkdist` tool must reside in its originally installed location within the SEI Toolkit. When `mkdist` is invoked to generate a SPECTRUM distribution medium from an existing manufacturing area, the current directory must be the root directory of the manufacturing area (as specified via the `$MANF` environment variable when `mkmanf` was used to create the manufacturing area).

Command Line Options:

The following options are available for the `mkdist` tool:

`check=min` Minimal dependency checking. Ignore extension module dependency errors that

would otherwise prevent generation of the distribution medium.

check=print Print dependency problems of all extension modules in the manufacturing area. Do not create a distribution medium.

Do not use the **check=min** and **check=print** options together on the mkdist command line. If neither **check=min** nor **check=print** options are specified, `mkdist` checks extension module dependencies and fails if dependency problems are detected.

clobber=y (Default) When generating a vtape, overwrite the specified vtape if it already exists.

clobber=n When generating a vtape, fail if the specified vtape already exists.

comp=y (Default) Check extension module dependencies specified by index file **comp:** entries. Overridden by **check=min**.

comp=n Do not check extension module dependencies specified by **comp:** index file entries.

dep=y Check extension module dependencies specified by index file **dep:** entries. Overridden by **check=min**.

dep=n (Default) Do not check extension module dependencies specified by index file **dep:** entries.

VAR extension module developers should not use **comp=y**, **comp=n**, **dep=y** or **dep=n** options on the `mkdist` command line.

level=<level>
Copy all extension modules of level <level> in the manufacturing area onto the distribution medium. <level> must be one of **0**, **1**, or **2**. The extension module level is specified by the **level:** entry in the extension module index file. Placing all three options (**level=0**, **level=1**, and **level=2)** on the `mkdist` command line results in copying all extension modules in the manufacturing area to the distribution

medium.

plist=<file>
Use parts list file <file> to specify which extension modules to copy to the distribution medium. <file> must be specified by base name (the filename without the extension) and must exist in the manufacturing area root directory. <file> must contain a list of names of extension modules to be copied to the distribution medium, one name per line.

<file> Same as **plist=<file>**. Matches any argument not recognized as another option.

prod=<product>
Copy only those components of each extension module associated with the SPECTRUM product <product> to the distribution medium. <product> must be one of **SS** (for SpectroSERVER) or **SG** (for SpectroGRAPH). This option applies to level 0 and 1 extension modules only. mkdist always copies all components of level 2 extension modules to the distribution medium.

If neither the **prod=SS** nor **prod=SG** option is specified, all components of all specified extension modules are copied to the distribution medium.

remmm=y If an extension module is missing from the manufacturing area, remove it from any extension module groups listed in the Management Module Master List.

remmm=n (Default) If an extension module is missing from the manufacturing area, remove any extension module groups that contain it from the Management Module Master List.

The **remmm=y** option is recommended when generating distribution media that is to be installed and tested at the local development site. **remmm=y** allows developers to install and test single extension modules without having to install the entire extension module groups to which they belong.

remmm=y should not be used when generating distribution media for installation by end users, because it allows installation of extension module groups that do not contain all of their specified extension modules.

vtape=<dir> Generates a virtual tape (vtape) distribution medium in directory <dir>. Virtual tapes are discussed in the section titled *Virtual Tape (Vtape)*, on page 4-29.

This argument formerly was an alternate to the **tape=<dev>** option. Since support for magnetic tape has been retired, this argument is now the only distribution format supported directly by `mkdist`, which will fail if this argument is not given.

An example of the use of the **vtape=<dir>** option is given in the section titled *Vtape Creation Example*, on page 4-29.

Input Files: The `mkdist` tool uses the following input files:

Parts list file

When the **plist=<file>** or **<file>** option is given, `mkdist` reads a parts list file in the manufacturing area root directory. The parts list file contains a list of extension modules to be copied to the distribution medium. The parts list file is formatted with one SPECTRUM extension module name per line, such as the following:

```
SMTE-LAN
SMBDG-CSI
SMHUB-CSI
```

You can produce a parts list file called `parts` that contains all the extension modules in the manufacturing area by using `mkarea` with the `parts` option, or you can create a parts list file using a text editor.

Manufacturing area files

The `mkdist` tool reads files in the manufacturing area and adds them to the distribution medium.

Output Files: The `mkdist` tool generates the following output files:

`Virtual tape`
When the **`vtape=<dir>`** option is in effect, `mkdist` creates files in the vtape directory `<dir>`.

`Distribution log file`
`mkdist` creates a distribution log file within the manufacturing area. This file is useful for detecting errors that may have occurred during generation of a distribution medium.

Exit Status: If `mkdist` is able to create the specified SPECTRUM distribution medium, it returns with exit status `0`. If mkdist cannot successfully create the medium for any reason, it returns with exit status `1`.

Errors: `mkdist` can generate error messages similar to the following:

```
Errors detected in manufacturing area:
Warning: MMML PP SA-CSI1000 MM CORE not in Manf area.
Warning: Removing MM CORE from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM Ibm6611 not in Manf area.
Warning: Removing MM Ibm6611 from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM NODELS not in Manf area.
Warning: Removing MM NODELS from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM Notifier not in Manf area.
Warning: Removing MM Notifier from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM OBackup not in Manf area.
Warning: Removing MM OBackup from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM SG-Support not in Manf area.
Warning: Removing MM SG-Support from MMML PP SA-CSI1000.
...
```

These error messages can be addressed by updating the MMML in the manufacturing area, as described in *Updating the MMML File*, on page 4-18.

Retired Features:

The following ***mkdist*** features, prominent in earlier versions of the SEI Toolkit, were retired as of SPECTRUM 4.0:

`cleanup` This command line option was formerly used to control cleanup of temporary files in the manufacturing area. This functionality has been retired.

core This command line option has been replaced by the **level=0** option. (*See also* **mm** and **xtn**.)

link=y, **link=n** These command line options were formerly used to control creation of vtapes in which records were symbolic links to records in the manufacturing area. This functionality has been retired.

log=y, **log=n**, **nolog**
These command line options were formerly used to control creation of a tape log record on a SPECTRUM distribution medium. This functionality has been retired.

mm This command line option has been replaced by the **level=1** option. (*See also* **core** and **xtn**.)

prompt=y, **prompt=n**, **prompt**, **verbose**
These command line options were formerly used to control user interactivity with `mkdist`. This functionality has been retired.

remote=y, **remote=n**, **remotetape**
These command line options were replaced in Rev. 4.0 by the **rhost=<host>** option, which was itself retired for Rev. 5.0 in favor of CD-ROM distribution.

SpectroGRAPH This command line option has been replaced by the **prod=SG** option.

SpectroSERVER This command line option has been replaced by the **prod=SS** option.

verify=y, **verify=n**
These command line options were formerly used to control validation of distribution medium contents. This functionality has been retired.

vtape <dir> This command line option has been replaced by the **vtape=<dir>** option.

x This command line option was formerly used to control distribution of X Windows software on SPECTRUM distribution media. This functionality has been retired.

xtn This command line option has been replaced by the **level=2** option. (*See also* **core** and **mm**.)

rhost=<host> This option was used to generate a physical distribution medium (QIC-150 tape) using a device (tape drive) attached to the remote host machine `<host>`. Support of magnetic tape has been dropped in favor of CD-ROM distribution.

tape=<dev> Generated a physical SPECTRUM distribution medium, using the tape drive device attached to device driver <dev>. Support of magnetic tape has been dropped in favor of CD-ROM distribution.

SPECTRUM Distribution Media

VAR developers can distribute SPECTRUM software on two types of media:

- **Virtual tapes**, also called **vtapes** — which are made up of files on disk.
- CD-ROM virtual tapes (for Windows NT platforms, only), also called CD-ROM vtapes — physical CD-ROMs that can be transported to a customer site and installed using a CD-ROM drive. The CD-ROM vtape format is *not the same* as the "official" CD-ROM format used by Cabletron to ship and install official SPECTRUM release software; that official format is not available to VAR developers.

Every SPECTRUM distribution medium created by using the tools in the SEI Toolkit will contain all the necessary installation support software, including the `Install` installation tool. The ***SPECTRUM Installation Guide*** contains directions for installing software from each supported type of SPECTRUM distribution medium.

The following sections describe each type of SPECTRUM distribution medium, along with examples of their creation and installation.

Virtual Tape (Vtape)

Creation and installation of virtual tapes (also called vtapes) is supported on all SPECTRUM platforms.

A vtape is made up of disk files, formatted so that they can be installed with the `Install` installation tool. Because they reside on disk, the creation and installation of these files do not require the use of special hardware devices, such as tape drives or CD-ROM drives. This makes vtapes ideal for use for generating and installing SPECTRUM extension module software in a development/test environment.

Vtape Creation Example

Suppose, for the purposes of demonstration, that you have the following items:

- The SEI Toolkit, installed in the directory `/usr/Spectrum/INSDK`.

- A Solaris manufacturing area, located within the directory `/usr/Spectrum/manfarea/sun4c.dist`.
- A parts list file called `/usr/Spectrum/manfarea/myparts`.

Suppose further that you wish to create a virtual tape in the directory `/usr/Spectrum/tapedir/vtape`, which contains the extension modules listed in the `myparts` file.

To create the vtape, you must invoke the following commands:

```
cd /usr/Spectrum/manfarea
/usr/Spectrum/INSDK/mkdist plist=myparts \
vtape=/usr/Spectrum/tapedir/vtape
```

When `mkdist` finishes, the vtape directory, `/usr/Spectrum/tapedir/vtape`, will contain a virtual tape, in the form of virtual tape records (files) with names of the form `vtape.<num>`, where `<num>` is a decimal record number. You can install the virtual tape from its present location, or you can copy it to another location, or you can connect to it through an NFS mount on another host of the same platform type and install it there. The ***SPECTRUM Installation Guide*** provides the detailed procedure for installing a virtual tape.

When copying vtapes, or when NFS-mounting virtual tapes, make sure that the name of the bottom directory level in the vtape directory is the same as the unextended names of the records contained in the vtape directory. For example, a vtape in the directory `/usr/Spectrum/tapedir/vtape` *must have records named* `vtape.0`, `vtape.1`, `vtape.2`, *etc. Otherwise,* `Install` *will fail to install the vtape.*

CD-ROM Virtual Tape (CD-ROM Vtape)

(Windows NT Only)

On the Windows NT platform, VAR-developed SPECTRUM software may be shipped and installed from CD-ROM vtape. A CD-ROM vtape is effectively a vtape on an CD-ROM.

The CD-ROM vtape format is *not the same* as the "official" CD-ROM format used by Cabletron to ship and install official SPECTRUM release software. CD-ROM vtapes are far easier to create and simpler to install than official Cabletron CD-ROMs, in large part because they do not have to support complicated security features required by official Cabletron CD-ROMs. The official Cabletron CD-ROM format is not currently available for use by VAR developers.

CD-ROM Vtape Creation Example

Suppose, for the purposes of demonstration, that you wish to create a SPECTRUM distribution on CD-ROM vtape on the Windows NT platform (the only platform on which this may be done at this time). Suppose that you have the following items:

- The SEI Toolkit installed in the directory `C:/win32app/Spectrum/INSDK`.
- A Windows NT manufacturing area, located within the directory `C:/win32app/Spectrum/manfarea/intel`.
- A parts list file, named `C:/win32app/Spectrum/manfarea/myparts`.
- A compact disk recorder (CDR).
- A recordable CD-ROM.
- Pre-mastering and mastering software for formatting and recording data on CD-ROM, conforming to the ISO 9660 CD-ROM format standard. Cabletron does not provide this software, but it is commercially available.

Suppose further that you wish to create a SPECTRUM distribution on the recordable CD-ROM, with that distribution to contain the extension modules listed in the `myparts` file. To do this, you must do the following:

1. Create a vtape containing the SPECTRUM extension modules listed in the `myparts` file. You can do this by following the example in *Vtape Creation Example*, on page 4-29 (modify this example to use the appropriate directory names). Make sure that you create the virtual tape in a directory whose lowest level is named `vtape`, such as `C:/win32app/Spectrum/vtape`.

2. Create a CD-ROM image file of the vtape that can be placed on CD-ROM. This step is called *pre-mastering*, and is done through the use of commercially available software. Cabletron does not provide pre-mastering software.

   Make sure that the virtual tape's `vtape` directory level is placed in the root directory within the CD-ROM image file. The goal is that the resulting CD-ROM vtape should contain the top-level directory `/vtape`, which in turn contains the vtape records `vtape.0`, `vtape.1`, etc.

3. Write the CD-ROM image file to the CD-ROM. This step is called *mastering*. During the mastering process, the CD-ROM recorder physically writes the CD-ROM image file to the recordable CD-ROM. After mastering, the CD-ROM is *fixed*, thereby making the CD-ROM information accessible. Mastering and fixing are done by commercially available mastering software, which is not supplied by Cabletron.

When mastering and fixing are done, the resulting recordable CD-ROM is now a CD-ROM vtape. It can be physically transported to another same-platform host having an attached CD-ROM drive and then can be installed on that

host. The ***SPECTRUM Installation Guide*** provides the procedure for installing a CD-ROM vtape.

Integration Tools Specification

This section describes the tools needed to complete a successful integration of SPECTRUM-compatible extensions with an installed SPECTRUM system. Refer to the ***SPECTRUM Installation Guide*** for more details.

Integration: Install

Usage: Refer to the ***SPECTRUM Installation Guide***.

Description: The `Install` program is not part of the SEI Toolkit, and is only being mentioned here for completeness.

The `Install` program installs SPECTRUM extension modules by extracting extension module files from the distribution medium and performing any necessary processing of the extracted files. `Install` provides a sophisticated graphical user interface, which allows the user to interactively select extension modules for installation.

For more detailed information on `Install`, refer to the ***SPECTRUM Installation Guide.***

Integration: Install.quick

Usage: **`./INSDK/Install.quick <arglist>`**

Description: `Install.quick` is designed to speed up the process of testing extension module software in the extension module development environment. After an extension module developer has used `mkmm` to create one or more extension module archives, `Install.quick` may be used to install the extension module software directly into an existing installed SPECTRUM area without having to manually perform the steps (`mkmanf`, `mkarea`, `mkdist`, and `Install`) normally required to distribute and integrate extension modules.

`Install.quick` looks in the current directory for all files with the extension `.tar`, which it takes to be archives for the extension modules to be tested. It invokes `mkmanf`,

`mkarea`, and `mkdist` to create a temporary manufacturing area and virtual tape containing just these extension modules. It then invokes `Install` to install the temporary virtual tape in the current directory. Finally, it removes the temporary virtual tape.

System Environment:

`Install.quick` must reside in its installed location within the SEI Toolkit.

`Install.quick` must be invoked by the Target Ownership user specified when SPECTRUM was installed (Administrator on NT hosts).

If the test extension module(s) are dependent on other extension modules, then `Install.quick` must be invoked in the root directory of an existing SPECTRUM installation area containing all extension modules on which the test extension module depends; this is always the case with level 0 and level 1 extension modules. If the test extension module(s) are not dependent on other extension modules, `Install.quick` may be invoked in an empty directory; in fact, this is normally the case with level 2 extension modules.

Command Line Options:

<optlist> List of command line options that are legal on the command line of the Install program, except for the following options:

device_name=<value>
install_type=<value>
media_host=<value>
media_type=<value>
sg_install=<value>
sg_link=<value>
ss_install=<value>
ss_link=<value>
xtn_install=<value>

It will seldom if ever be necessary to use any options on the `Install.quick` command line. For detailed information on legal `Install` command line arguments, refer to the ***SPECTRUM Installation Guide***.

Input Files: The `Install.quick` tool uses the following input files:

Extension module archives
`Install.quick` reads and installs any extension module archives existing in the root directory of the SPECTRUM

installation area (current working directory).

Output Files: The ***Install.quick*** generates the following output files:

`Installed files`

`Install.quick` installs files into the SPECTRUM installation area. The exact set of files created and modified depends on the extension modules that are being installed.

Exit Status: If `Install.quick` is able to install the extension module archive into the SPECTRUM area, it returns with exit status `0`. If the installation cannot be successfully completed for any reason, `Install.quick` returns with exit status `1`.

Errors: `Install.quick` can generate warning messages similar to the following:

```
Errors detected in manufacturing area:
Warning: MMML PP SA-CSI1000 MM CORE not in Manf area.
Warning: Removing MM CORE from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM Ibm6611 not in Manf area.
Warning: Removing MM Ibm6611 from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM NODELS not in Manf area.
Warning: Removing MM NODELS from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM Notifier not in Manf area.
Warning: Removing MM Notifier from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM OBackup not in Manf area.
Warning: Removing MM OBackup from MMML PP SA-CSI1000.
Warning: MMML PP SA-CSI1000 MM SG-Support not in Manf area.
Warning: Removing MM SG-Support from MMML PP SA-CSI1000.
...
```

These warning messages do not adversely affect the behavior of `Install.quick`, however, and a future version of the SEI Toolkit will address the causes of these messages so as to prevent confusion.

Chapter 5

Shippable Files

This chapter explains the concepts of levels, products, and file types, and how they affect distribution and installation of files shipped in SPECTRUM extension modules. This discussion also includes reference tables describing files shippable in the SPECTRUM extension module, as well as detailing the development and distribution of certain specific shippable files.

Introduction

SPECTRUM extension modules support distribution and installation of many kinds of shippable files. Files of different kinds are associated with different SPECTRUM products, and these different types must be developed, distributed, and installed in different ways. This chapter covers the basic concepts (levels, products, and file types) used within the SEI Toolkit to classify shippable files for appropriate distribution and installation processing. The chapter outlines the development, distribution, and installation of every supported type of shippable file.

Two types of files are specific to the SEI Toolkit: *MM description files* and *custom install scripts*. These files facilitate the proper installation of extension modules. Another type of file, the X resource file template, is used in many SPECTRUM graphical products. This chapter details the development, distribution, and installation of these file types, as well as providing notes on the development of other file types.

Levels, Products and File Types

The way in which a shippable file in an extension module is packaged, distributed, and installed is determined largely by the following factors:

- ***The level associated with the extension module containing the file*** — The level is specified via the "**`level: <level>`**" entry in the index file for the extension module, and applies to every file in the extension module. The value of `<level>` may be any one of the digits **`0`**, **`1`**, or **`2`**.

- ***The product associated with the file —*** The product is specified in the <prod> field of the index file's **file:** or **head:** entry used to ship the file. This <prod> field may have any one of the values **Install**, **SS**, or **SG**.
- ***The file type of the file —*** The file type is specified in the <type> field of the **file:** entry used in the index file that ships the file. The type may take on any one of many values, as documented later in this chapter.
- ***The base name extension of the shipped file —*** The base name extension is specified in the <tname> field of the index file's **file:** entry or in the **head:** entry used to ship the file.

Together, the level and product of a file determine its general relationship to an installed SPECTRUM product, as outlined in the following table:

Table 5-1. Relation of File to Installed SPECTRUM Product by Index File Level and Product

<level>	<prod>	Identification and Relation to Installed Product
0, 1, 2	Install	Install support file. Needed to support the SPECTRUM Install program. Install support files also include custom install scripts (files of type cus) and files shipped via the **head:** entry, even if the file product is not Install.
0	SS	SpectroSERVER core file. Required for installation of the SpectroSERVER product. Level 0 extension modules are produced only by Cabletron.
1	SS	SpectroSERVER management module file. May be optionally integrated into the SpectroSERVER product (typically, to help SpectroSERVER manage a particular device or device group).
2	SS	SpectroSERVER external application file. Part of a separate application or tool used in conjunction with SpectroSERVER, but not integrated into SpectroSERVER itself. A Level 2 extension module can be thought of as belonging to its own standalone product.
0	SG	SpectroGRAPH core file. Required for installation of the SpectroGRAPH product. Level 0 extension modules are produced only by Cabletron.
1	SG	SpectroGRAPH management module file. May be optionally integrated into the SpectroGRAPH product (typically to help SpectroGRAPH display a particular device or device group).
2	SG	SpectroGRAPH external application file. Part of a separate application or tool used in conjunction with SpectroGRAPH, but not integrated into SpectroGRAPH itself. A Level 2 module can be thought of as belonging to its own standalone product.

Development of Shippable Files

Table 5-2 outlines preferred development methods for files that are to be shipped in SPECTRUM extension modules. Each row of the table lists:

- ***The product of the shippable file*** — as specified in the `<prod>` field of the **`file:`** or **`head:`** entry in the extension module index file.
- ***The file type of the shippable file*** — as specified in the `<type>` field of the **`file:`** or **`head:`** entry in the extension module index file.
- ***A short description of the shippable file of the given product and type*** — identifying the applicable nomenclature as a descriptive name.
- ***A short description of the format of the shippable file*** — indicating the preferred method of development for the file where possible. Where special editors are recommended, the table gives references.

Table 5-2. Preferred Development Methods for Shippable Files

<prod>	<type>	Description	File Format
`Install`	`doc`	Install documentation file	Text file.
`Install`	`file`	General `Install` file	Any permissible format.
`Install`	`mmdesc`	MM description file	Text file. (See *MM Description Files*, on page 5-11 for more information.)
`Install`	`tool`	`Install` tool file	Compiled program, script, or data file.
`SG`	`appdef`	SG X resource (app-defaults) template file	X resource file with embedded environment variables. (See *X Resource File Templates*, on page 5-26, for more information.)
`SG`	`cus`	SG custom install script	Bourne shell script fragment. (See *Custom Install Scripts*, on page 5-13 for more information.)
`SG`	`doc`	SG documentation file	Text file.
`SG`	`file`	SG general file	Any permissible format.
`SG`	`gib`	SG Graphical Information Block (GIB) file	Text file in SG GIB file format. Can be created and edited using the GIB Editor, or manually. (For more on the GIB Editor, see the ***SPECTRUM GIB Editor Guide***.)
`SG`	`icon`	SG icon object file	Compiled object file.
`SG`	`iconmap`	SG model type to icon map file	Text file, edited manually. (For more on icon mapping, refer to the ***SPECTRUM IIB Editor Guide***.)
`SG`	`iib`	SG Icon Information Block (IIB) file	Text file in SG IIB file format. Can be created and edited using the IIB Editor, or manually. For more on the IIB Editor, see the ***SPECTRUM IIB Editor Guide***.)

Table 5-2. Preferred Development Methods for Shippable Files (Continued)

<prod>	<type>	Description	File Format
SG	image	SG image file	Data file. Can be created and edited using the `csiedit` tool from the Imaging Toolkit. (For more on `csiedit`, see the ***Imaging Toolkit Guide***.)
SG	isvmap	SG model type to action description map file	Text file, edited manually. (For more on action description mapping, refer to the ***SPECTRUM IIB Editor Guide***.)
SG	lib	SG library file	Compiled object library.
SG	obj	SG general object file	Compiled object file.
SG	other	SG miscellaneous data file	Text files in various formats, including: • Event files and probable cause **`files:`** Text files created and edited via the Control Panel, or manually. For more on Control Panel, see ***SPECTRUM Control Panel Online Help***. • Script files: Scripts, created and edited manually. • Sound files: Data files, not editable, format internal to Cabletron.
SG	pib	SG PIB contribution file	Text file in PIB contribution file format. Created and edited manually. See *PIB Contribution Files*, starting on page 5-30, for more information.
SG	rib	SG Report Information Block (RIB) file	Text file in SG RIB file format. Can be created and edited using the RIB Editor, or manually. For more on RIB files, refer to the ***SPECTRUM Report Generator User's Guide***.
SG	spmamap	SPECTRUM Portable Management Application Map	Text file. Undocumented format internal to Cabletron. See *stdmenu and spmamap files* section on page 5-30 for more information.
SG	stdmenu	SG standard menu file	Fragment of X resource file with embedded environment variables. Created and edited manually. See *stdmenu and spmamap files* section on page 5-30 for more information. For more on stdmenu files and their contents, refer to the ***SPECTRUM Partner's Guide to Integrating Applications***.
SG	tool	SG tool file	Compiled program, script, or data file.
SG	tplinc	SG PGUI template include file	Text file. Undocumented format, internal to Cabletron.
SG	view	SG view object file	Compiled object file.
SG	xtn	SG external application file	Any permissible format.
SS	appdef	SS X resource (app-defaults) template file	X resource file with embedded environment variables. See *X Resource File Templates* section on page 5-26 for more information.
SS	convobj	SS database partition object file	Compiled object file.

Table 5-2. Preferred Development Methods for Shippable Files (Continued)

<prod>	<type>	Description	File Format
SS	convtab	SS database partition conversion table	Text file. Undocumented format, internal to Cabletron.
SS	cus	SS custom install script	Bourne shell script fragment. See *Custom Install Scripts* section on page 5-13 for more information.
SS	db	SS database import file	Data file. Format created by "dbtool export". (For more information on dbtool, refer to the ***SPECTRUM Model Type Editor User's Guide***.)
SS	doc	SS documentation file	Text file.
SS	file	SS general file	Any permissible format.
SS	ih	SS inference handler object file	Compiled object file.
SS	lib	SS library file	Compiled object library.
SS	mi	SS model intelligence object file	Compiled object file.
SS	obj	SS general object file	Compiled object file.
SS	tool	SS tool file	Compiled program, script, or data file.
SS	vfile	SS vendor file	Text file in Alertmap file format. (For more on Alertmap file format, refer to the ***SPECTRUM Basic Extension Guide***.)
SS	xtn	SS external application file	Any permissible format.

Distribution of Shippable Files

Table 5-3, following, outlines default distribution parameters for files that can be shipped in SPECTRUM extension modules. Each row of the table lists:

- ***The product of the shippable file*** — as specified in the <prod> field of the **file:** or **head:** entry in the extension module index file.
- ***The file type of the shippable file*** — as specified in the <type> field of the **file:** or **head:** entry in the extension module index file.
- The standard default target directory into which Install extracts the shippable file, relative to the installation root directory. This directory is defined in the mkmm***.sys*** data file that resides in the same directory with mkmm.
- Special notes on use of the file type (provided at the end of the table).

As of SPECTRUM 4.0, the standard default source directory (<sdir>) for all file types became "." (the directory in which mkmm is invoked). To restore the pre-4.0 <sdir> defaults for various file types, you can add certain lines to the mkmm***.loc*** file, as described under the discussion of $ext and $vend under mkmm (see page 4-10).

To include a shippable file of product <prod> and type <type> in an extension module, include a "**file: <prod> <type>**" entry in the index file for the extension module.

Table 5-3. File Statistics by Product and Type

<prod>	<type>	<tdir>	Note
Install	doc	Readme	1
Install	file	.	1
Install	mmdesc	Install-Tools/MMD	1
Install	tool	Install-Tools	1
SG	appdef	SG-Tools/appdef	
SG	cus	.	1
SG	doc	Readme	
SG	file	SG	
SG	gib	SG-Support	
SG	icon	SG	2
SG	iconmap	SG-Tools/iconmap	
SG	iib	SG-Support	
SG	image	SG-Support	
SG	isvmap	SG-Tools/isvmap	
SG	lib	SG	2
SG	obj	SG	2
SG	other	SG-Support	
SG	pib	SG-Tools/pib	
SG	rib	SG-Support	
SG	spmamap	SPMAL/defs	2
SG	stdmenu	app-defaults/def	

Table 5-3. File Statistics by Product and Type (Continued)

<prod>	<type>	<tdir>	Note
SG	tool	SG-Tools	
SG	tplinc	SG-Tools/tplinc	
SG	view	SG	2
SG	xtn	.	
SS	appdef	SG-Tools/appdef	
SS	convobj	SS-Tools/dbpart1	
SS	convtab	SS-Tools/dbpart1	
SS	cus	.	1
SS	db	SS	
SS	doc	Readme	
SS	file	SS	
SS	ih	SS	
SS	lib	SS	2
SS	mi	SS	
SS	obj	SS	2
SS	tool	SS-Tools	
SS	vfile	SS	2
SS	xtn	.	

Notes:

1. Files of product `Install` or file type `cus` are always shipped in the first record (`Install` record) of the distribution medium (virtual tape or CD-ROM). This is true even if a **`file:`** entry, as opposed to a **`head:`** entry, is used to ship the file.
2. Files of this product and file type are reserved for use in Cabletron-produced extension modules. They should not be shipped in VAR-developed extension modules.

Installation of Shippable Files

To install a usable SPECTRUM extension module, it is seldom enough to extract the files from the medium to the correct location in the installation area. Additional processing of the installed files typically needs to be done—for example, linking object files into executables, or importing SpectroSERVER database import files into the SpectroSERVER database.

The `Install` program performs much of this additional processing as standard installation services. `Install` decides what processing to perform on a shippable file by looking at the product and file type of the file. For example, files of product "SS" and file type "db" are identified as database import files and imported into the SpectroSERVER database.

If a file is erroneously shipped in an extension module of inappropriate level, `Install` may not correctly process the file.

Similarly, if a file is erroneously shipped with a base name extension incompatible with its file type, `Install` may not correctly process the file.

The following table outlines the standard installation processing performed on files of each supported product and file type. Each row of the table lists:

- ***The file type of a shippable file —*** as specified in the <type> field of the `file:` or `head:` entry in the extension module index file.
- ***The appropriate levels for extension modules used to ship the file —*** as specified via the `level:` entry of the extension module index file.
- ***The appropriate base name extensions for the shipped file (if applicable) —*** as specified in the <tname> field of the `file:` or `head:` entry.
- ***A summary of the standard processing that*** `Install` ***performs on the file.***

Table 5-4. Special Processing of Shippable Files by Type

Type	Level	Base Name	Standard *Install* Processing Performed
appdef	0, 1, 2	N/A	`Install` expands environment variables in X resource file templates to produce customized X resource files for graphical applications. For more on writing and using X resource file templates, see the *X Resource File Templates* section on page 5-26.
convobj	0, 1	*.o	Installation of database partition conversion object file will cause SpectroSERVER and Database partition converter executables to be linked.

Table 5-4. Special Processing of Shippable Files by Type (Continued)

Type	Level	Base Name	Standard *Install* Processing Performed
convtab	0, 1	*.v	Installation of database partition conversion table will cause SpectroSERVER and Database partition converter executables to be linked.
cus	0, 1, 2	*.cus	In the index file's file: or head: entry, **<prod>=Install** and **<type>=cus** should not be used together. `mkmm` adds function definitions to the custom install script and makes it executable before committing it to the MM archive file. `Install` invokes custom install scripts after extraction of all MM records. A management module should use such a custom install script for special install-time processing of files it distributes to its associated MM record. `Install` archives and deletes custom install scripts prior to terminating. For more on writing and using custom install scripts, see the *Introduction* section on page 5-1.
db	0, 1	*.e	`Install` imports database import files into the SpectroSERVER database. `Install` archives and deletes database import files prior to terminating.
doc	0, 1, 2	N/A	No special installation processing.
file	0, 1, 2	N/A	No special processing is performed on general files. Any file that does not need special `Install` processing should be shipped using this file type.
gib	0, 1	N/A	Before extraction of any MM records, `Install` evaluates checksums of previously installed GIB files to detect user modifications. User-modified GIB files are saved to protect them from overwrite during extraction.
icon	0, 1	*.o	`Install` links icon object files into the SpectroGRAPH executable. `Install` archives and deletes icon object files prior to terminating.
iconmap	0, 1	*.map	`Install` uses iconmap files to build a data file for the Portable Graphical User Interface (PGUI) used by Level 2 Toolkits.
ih	0, 1	*.o	`Install` links IH object files into the SpectroSERVER executable. `Install` archives and deletes IH object files prior to terminating.
iib	0, 1	N/A	Before extraction of any MM records, `Install` evaluates checksums of previously installed IIB files to detect user modifications. User-modified IIB files are saved to protect them from overwrite during extraction.
image	0, 1	N/A	No special installation processing.
isvmap	0, 1	*.map	`Install` uses `isvmap` files to build a data file for the Portable Graphical User Interface (PGUI) used by Level 2 toolkits.
lib	0, 1	*.a	`Install` links library files into the SpectroSERVER or SpectroGRAPH executable.

Table 5-4. Special Processing of Shippable Files by Type (Continued)

Type	Level	Base Name	Standard *Install* Processing Performed
mi	0, 1	*.o	`Install` links MI object files into the SpectroSERVER executable. `Install` archives and deletes MI object files prior to terminating.
mmdesc	0, 1, 2	*.mmd	`Install` displays the contents of the MM description file in the MM description panel of the MM selection window when the MM is selected for installation. For more on writing and shipping MM description files, see the *Introduction* section on page 5-1.
obj	0, 1	*.o	`Install` links general object files into the executable associated with the product <prod>. `Install` archives and deletes general object files prior to terminating.
other	0, 1	N/A	No special installation processing.
pib	0, 1	*.p	`Install` uses information in PIB contribution files to generate the `CsViewControl` file and PIB files used by SpectroGRAPH.
rib	0, 1	N/A	No special installation processing.
spmamap	0, 1	*.map	`Install` expands environment variables in `spmamap` files, and uses them to build an input file for the generic application launcher.
stdmenu	0, 1	*.o	`Install` expands environment variables in `stdmenu` files, and uses them to build the `CsStdMenu` menu configuration file used by SpectroGRAPH. For more on stdmenu files, refer to the ***SPECTRUM Partner's Guide to Integrating Applications***.
tool	0, 1, 2	N/A	No special installation processing.
tplinc	0, 1	N/A	`Install` uses `tplinc` files to build a data file for the Portable Graphical User Interface (PGUI) used by Level 2 Toolkits.
vfile	0, 1	N/A	No special installation processing.
view	0, 1	N/A	`Install` links view object files into the SpectroGRAPH executable. `Install` archives and deletes view object files prior to terminating.
xtn	2	N/A	External application files should be shipped in Level 2 MMs only, never in Level 0 or Level 1 MMs. This is the default file type used for shipping files in Level 2 MMs. The presence of an external applications file in a Level 0 or Level 1 extension module will cause `Install` to link executables for the associated product.

Special Shippable Files

The remainder of this chapter details the development, distribution, and installation of certain special types of shippable files, including:

- ***MM description files —*** which let the extension module developer include a detailed description of the extension module to help the installer select the appropriate extension module to install.
- ***Custom install scripts —*** which let the extension module developer perform special installation operations not otherwise provided by the `Install` program.
- ***X resource file templates —*** which are used to install site-specific X resource files (app-defaults files) for various SPECTRUM graphical applications.

This discussion also includes special notes concerning the development of other file types. Additional information is provided in Chapter 6, *Use Scenarios*, which also shows how developers can ship modules via magnetic tape.

MM Description Files

Introduction

At installation time, the `Install` program lets the installing user select extension modules for installation via a graphical interface that is shipped with the extension module in an MM description file. This interface displays a short description of each extension module to aid the user in the selection.

The `mkmm` utility requires each shippable extension module to include an MM description file; if the index file does not contain an entry to ship an MM description file, `mkmm` will fail to generate the extension module archive.

Writing MM Description Files

An MM description file is easy to write. Suppose, for the sake of example, that you have an index file named `FOO.i`. In the same directory that contains that `FOO.i` file, you would create an MM description file called `FOO.mmd` (the `.mmd` extension is required). `FOO.mmd` should contain one or two paragraphs describing the extension module. The following text is a representative example of what you might say:

```
The FooCorp Route Obliterator Management Module supports model
types for the standalone FooCorp Route Obliterator device
(ObFoo9000) and for the FooCorp Route Obliterator Media Interface
Module (ObFoo9000MIM), which provides the same functionality as
the standalone unit when installed in a Cabletron MMAC hub
chassis.
```

If you are developing under source control, add the `FOO.mmd` description file to your source control system.

Shipping MM Description Files

An MM description file is used to help the user select the extension module for installation. To ensure that the description file is present at the installation site, the index file for a particular MM index file includes entries for shipping the MM description file along with the other entries. The following representative `FOO.i` index file, for example, includes the entry "**`file: Install mmdesc ? FOO.mmd ? ?`**" in order to distribute `FOO.mmd` as an MM description file (the product name should always be "**`Install`**" for MM description files).

```
mm: FOO Management Module
vend: FOOCo
rev: 2.1rev0
irev: 02.01.00.000
level: 2
file: Install mmdesc ? FOO.mmd ? ?
[other "file:" entries follow]
```

Each index file should have *exactly one* MM description file shipped with it.

In general, the `file:` entry for shipping a MM description file should have the following fields:

<prod>	"**`Install`**"
<type>	"**`mmdesc`**"
<sdir>	directory containing MM description file (or "?" if the MM description file is in the same directory as the index file)
<sname>	basename of MM description file ("`FOO.mmd`" in the example)
<tdir>	"**?**"
<tname>	"**?**"

Custom Install Scripts

Introduction

The set of installation services provided by the `Install` utility is fairly complete, handling most extension module installation needs. Many extension modules need to perform special installation processing not provided by the `Install` utility, however. You can use custom install scripts to perform such specialized installation functions.

SPECTRUM software is installed in units called records. Each record is associated with a management module (MM) and a product (currently either "SS" for `SpectroSERVER` or "SG" for `SpectroGRAPH`). Each custom install script is in turn associated with a record.

The title "SS record for SMEPI-CSI," for example, would refer to a record associated with an SMEPI-CSI extension module and with the SpectroSERVER product. It follows that a custom install script associated with this record would be called an "SS custom install script for SMEPI-CSI." This custom install script would handle the special installation needs of the associated record.

A record *can* have two or more custom install scripts, but this is rarely necessary. Most records either have no associated custom install script or else have only one. Since a given extension module may have two associated records (an SS record and an SG record), however, it is not uncommon to find two custom install scripts associated with one extension module (an SS custom install script for the SS record, and an SG custom install script for the SG record).

You can include a custom install script in an extension module by adding a **`file:`** entry for it in the extension module index file (refer to *Writing Custom Install Scripts*, starting on page 5-16). Once you have added this entry, the SEI Toolkit distribution utilities (`mkmm`, `mkarea`, and `mkdist`) will ensure that the custom install script is properly distributed to the medium.

At the installation site, the custom install script is extracted as part of the `Install` record of the medium. For CD-ROM, the `Install` record is automatically extracted; for vtape distribution, it must be done manually. For detailed information on this process, refer to the ***SPECTRUM Installation Guide***.

If the `Install` utility is later instructed to install the extension module, it will first extract the extension module software from the medium and then invoke the custom install script to perform any special installation functions associated with that extension module.

The following discussion provides pointers on when to use custom install scripts and why to do so, gives details on writing these custom install scripts (with an extensive example), and explains (using that same example) how to

install these scripts. The discussion then concludes with a discussion of predefined Bourne shell variables (see page 5-20) and functions (see page 5-23) that you can use in these custom install scripts.

Uses of Custom Install Scripts

The `Install` utility provides standard services for the proper installation of many different types of files. Most extension modules can be properly installed using only these standard services, and these standard services should be used in preference to custom install scripts when possible.

However, `Install` cannot reasonably provide standard services for all of the many possible specialized installation functions that could conceivably be required by extension modules. When `Install` does not provide all the services needed to install a given extension module, custom install scripts should be used to help install the extension module. Since custom install scripts have all the capabilities of Bourne shell scripts, they can handle any special installation needs.

The following are examples of installation functions that should be handled using custom scripts:

- ***Special installation methods —*** The `Install` utility installation services are sufficient for most installation needs. Nevertheless, an extension module will sometimes need to perform special installation tasks unique to itself (such as interacting with third-party software). You can use custom install scripts to perform such specialized installation functions.
- ***Modifying contents of installed files to contain information specific to the installation site —*** Most of the time, the contents of a file, as extracted from the medium, do not need to be modified during installation. There are times, however, when installed files need to contain information specific to the installation site — such as the installation root directory or the installation host name, which cannot be known until installation time. It then becomes necessary to embed that information in the installed files.

  `Install` provides standard services for embedding installation-specific information in certain types of files (such as *X resource* files and *stdmenu* files) at installation time. `Install` does not provide such a service for all shippable files. If you must embed installation-specific information in a shippable file, and `Install` does not provide a service for doing this, you should use custom install scripts to embed the necessary information.
- ***Installing files into system locations —*** In order to protect the integrity of the installation host, the `Install` utility never modifies files in a system location, such as `/usr/bin` or `/usr/lib/X11`, without express permission of the installer. In particular, `Install` will not allow extraction of files into a system location.

If possible, the modification of files in system areas should be avoided during extension module installation. In some rare cases, however, modification to files in system locations is necessary for proper installation of an extension module. In these cases, custom scripts should be used to modify files in system locations. The custom script should notify the user when files in system locations are modified.

The following are prime examples of when *not* to use custom install scripts:

- ***Moving installed files from their location as extracted from the medium to some other location within the SPECTRUM installation area*** — When possible, avoid using custom scripts to move installed files from their extracted location to some other location. The **file:** and **head:** index file entries can be modified so that any shippable file can be extracted from the medium to any desired location in the SPECTRUM installation area. (For further details, refer to the *file:* discussion on page 3-18, and/or to the *fdir:* discussion on page 3-21.)

  In earlier versions of SPECTRUM media, custom scripts were commonly used to move groups of files from their locations as extracted from the medium to some other user-specified location during installation, allowing relocatable installation of products. This practice is now discouraged, since (1) installers rarely if ever choose to relocate installed SPECTRUM products, (2) `Install` cannot track files that have been relocated via custom scripts, and (3), a second installation can be used to install the product in the desired location if a SPECTRUM product actually needs to be relocated.

  In the event that a SPECTRUM product does need to be relocated, management of the relocated product must be handled entirely within the custom script, which will have the following responsibilities:

  - Relocating shippable files.
  - Obtaining, verifying, and tracking product locations.
  - Ensuring that subsequent installations of the product are properly integrated into the relocated product.

- **Updating PIB files.** - Do not use custom install scripts to modify SpectroGRAPH PIB files or the View Control file. Instead, use the "**`file: SG pib`**" entry to ship a PIB contribution file containing the desired PIB file updates. See *PIB Contribution Files*, starting on page 5-30, for more information.

- **Installing X resource files.** - Do not use custom install scripts to install X resource files for graphical applications. Instead, use the **`file: SG appdef`** entry for installing X resource files in compliance with SPECTRUM graphical application standards. See *X Resource File Templates*, starting on page 5-26, for more information.

Writing Custom Install Scripts

The following example provides information on how to write custom install scripts. The example, continuing the previous scenario, assumes you are developing the extension module FOO in the /usr/spectrum/devarea development area and that you have the following software files in the development area:

```
/usr/spectrum/devarea/src/foo_ss1.dat
/usr/spectrum/devarea/src/foo_ss2.dat
/usr/spectrum/devarea/src/foo_ss3.dat
/usr/spectrum/devarea/src/foo_sg1.dat
/usr/spectrum/devarea/src/foo_sg2.dat
/usr/spectrum/devarea/src/foo_sg3.dat
```

The example also assumes that, in order to package your software into the FOO extension module, you have created the index file /usr/spectrum/devarea/index/FOO.i, containing the following entries:

```
# Index file for FOO extension module
file: SS xtn /usr/spectrum/devarea/src foo_ss1.dat FOO_SS ?
file: SS xtn /usr/spectrum/devarea/src foo_ss2.dat FOO_SS ?
file: SS xtn /usr/spectrum/devarea/src foo_ss3.dat FOO_SS ?
file: SG xtn /usr/spectrum/devarea/src foo_sg1.dat FOO_SG ?
file: SG xtn /usr/spectrum/devarea/src foo_sg2.dat FOO_SG ?
file: SG xtn /usr/spectrum/devarea/src foo_sg3.dat FOO_SG ?
```

According to this index file, there are three files associated with the SS product. Therefore, the FOO extension module will have an SS record on the medium, containing the files:

```
FOO_SS/foo_ss1.dat
FOO_SS/foo_ss2.dat
FOO_SS/foo_ss3.dat
```

Similarly, the FOO extension module will have an SG record on the medium containing the files:

```
FOO_SG/foo_sg1.dat
FOO_SG/foo_sg2.dat
FOO_SG/foo_sg3.dat
```

At installation time, these files (three FOO_SS files and three FOO_SG files) will be extracted from the distribution media, relative to the installation root directory.

Suppose, further, that you wish to allow the user to move the files in FOO_SS to some other directory at installation time. Since you cannot know the identity of this to-be-selected directory in advance, you cannot modify the

index file such that the files will be extracted into their proper locations. To get around this problem, you can write a custom install script, such as `FOO.cus`, which will be shipped with the `FOO` extension module. When `Install` installs the `FOO` extension module, it will run `FOO.cus`, which in turn will let the user specify where the files are to be moved and also will move the files to the specified location.

For this example, you would create the custom install script, `/usr/spectrum/devarea/index/FOO.cus` (the `.cus` extension is required). Note that `FOO.cus` is in the same directory as `FOO.i`, as previously described. (Placing the custom script in the same directory with the index file is not required, but is convenient, because the `mkmm` utility by default looks for custom install scripts in that directory. You can change where `mkmm` locates the custom script by changing the `<sdir>` field of the `file:` entry that ships the custom script.)

Figure 5-1 shows how your final `FOO.cus` might appear (with line numbers added for your convenience with respect to the following explanations):

Figure 5-1 shows an example custom script for the task of moving files from their extracted location to a user-selected directory. We recommend that you do not write custom scripts for this purpose, however. This task was chosen as an example only because it is a relatively simple task that covers the important points of custom script writing.

Figure 5-1. FOO.cus Custom Install Script

```
 1  ##############################################################
 2  # SS custom install script for FOO extension module
 3  ##############################################################
 4
 5  # If Install is running in auto mode, you are not allowed
 6  # to ask the user for the target directory, so assume the
 7  # user does not want to move any files and exit happily.
 8  if [ "$AUTO" ] ; then
 9      echo "Not moving any files"
10      exit 0
11  fi
12
13  # Here Install is running in custom mode, and you may
14  # interact with the user. Ask the user for the name of
15  # the target directory where files are to be moved.
16  echo "Enter target directory for SS FOO files:"
17  read TARGETDIR
18
19  # If no target directory was specified, assume user did
20  # not want to move files and exit happily.
21  if [ ! "$TARGETDIR" ] ; then
22      echo "Not moving any files"
23      exit 0
24  fi
25
26  # Here the target directory was specified. The target
27  # directory may not exist, so create it. "mkdir" will
```

Figure 5-1. FOO.cus Custom Install Script (Continued)

```
28   # complain if the target directory already exists, so
29   # suppress "mkdir" errors.
30   mkdir -p "$TARGETDIR" 2>$DEV_NULL
31
32   # If we were unable to create the target directory, fail.
33   if [ ! -d "$TARGETDIR" ] ; then
34       echo "Could not create target directory $TARGETDIR"
35       exit 1
36   fi
37
38   # Here the target directory exists.  Copy the files to it.
39   cp -r $IROOT/FOO_SS/* $TARGETDIR
40
41   # Remove the original directory, and exit happily
42   rm -rf $IROOT/FOO_SS
43   echo "Done moving files"
44   exit 0
```

`FOO.cus` is fairly straightforward for developers familiar with the Bourne shell, but the following few points may not be readily apparent.

- `FOO.cus` is written in the Bourne shell (`sh`). All custom install scripts must be written in Bourne shell. A custom install script cannot be written in `Korn` shell (`ksh`) or `Perl`, for example, or as a compiled executable. You cannot circumvent this by using the "**`#!`**" convention (for example, you cannot use a "`#!/bin/ksh`" entry). The reason for this requirement is portability: the Bourne shell is universally available and fairly standard on all SPECTRUM installation platforms. Also, Bourne shell allows `Install` to provide portable installation support utilities.
- `FOO.cus`, like all custom install scripts, must be portable over those platforms on which it is liable to be invoked. Despite standardization efforts, however, a number of UNIX commands and options are not portable over all platforms supporting SPECTRUM installation. For instance, "**`grep -w`**" works on Solaris platforms, but it fails on NT platforms. Consequently, you would be wise to test any newly written custom install scripts on all relevant platforms.
- Line 8 of `FOO.cus` tests the `$AUTO` variable to see whether `Install` is running in auto (noninteractive) or custom (interactive) mode (refer to *Custom Install Script Predefined Environment Variables*, on page 5-20). In auto mode, the entire installation (and, in particular, the custom install scripts) should run without user intervention. If `FOO.cus` queried the user for the target directory in auto mode, the installation would stop at that point, awaiting user input. In auto mode, therefore, the program assumes the user doesn't want to move the files, and it exits the script immediately (line 10) with exit status 0 (zero) to indicate that the script exited successfully.
- On line 30 of `FOO.cus`, the `$DEV_NULL` variable is used instead of `/dev/null`. This is very important, because `FOO.cus` can potentially be

run on Windows NT hosts, where `/dev/null` is a plain file and `nul:` is the standard output sink device.

- On line 35, `FOO.cus` exits with exit status 1. This exit status tells `Install` that the `FOO.cus` failed. In response to the custom script failure, `Install` will discontinue installation of the `FOO` extension module and any other extension modules that depend on `FOO`. All custom install scripts should exit with a nonzero exit status in this same way if a fatal error is detected.
- On line 44, `FOO.cus` exits with exit status 0 (zero). The zero exit status tells `Install` that the custom script succeeded. All custom installation scripts should exit with status 0 upon successful completion. It is also recommended that you include "**`exit 0`**" as the last line of the custom script to ensure successful completion.

Note that `FOO.cus` does not contain the "**`##headersize`**" and "**`##include`**" custom script directives that were formerly used to include support functions in custom scripts (see the *Predefined Functions for Custom Install Script* discussion on page 5-23). These directives are no longer required, since every custom script now automatically includes all custom script support functions. The Bourne shell now treats "**`##headersize`**" and "**`##include`**" directives as comments, however, so there is no need to remove them from existing custom scripts.

Shipping Custom Install Scripts

Since custom install scripts perform special processing functions associated with the installation of a particular management module, as previously noted, they must be present at the installation site in order to perform those functions. To ensure this, you must include custom install script index file entries along with the other entries in an index file for a particular management module. In the example index file shown in Figure 5-2, index file `FOO.i` includes the entry "**`file: SS cus ? FOO.cus ? ?`**." This entry distributes `FOO.cus` as a custom install script for the "SS" product.

Figure 5-2. FOO.i Custom Install Script Index File (Fictitious)

```
mm: FOO Management Module
vend: FOOCo
rev: 2.1rev0
irev: 02.01.00.000
level: 2
file: SS cus ? FOO.cus ? ?
file: SS xtn /usr/spectrum/devarea/src foo_ss1.dat FOO_SS ?
file: SS xtn /usr/spectrum/devarea/src foo_ss2.dat FOO_SS ?
file: SS xtn /usr/spectrum/devarea/src foo_ss3.dat FOO_SS ?
file: SG xtn /usr/spectrum/devarea/src foo_sg1.dat FOO_SG ?
file: SG xtn /usr/spectrum/devarea/src foo_sg2.dat FOO_SG ?
file: SG xtn /usr/spectrum/devarea/src foo_sg3.dat FOO_SG ?
```

In general, the `file:` entry for shipping a custom script should have the following fields:

<prod>	the appropriate product ("SS" or "SG")
<type>	"**cus**"
<sdir>	directory containing custom script (or "**?**" if the custom script is in the same directory as the index file)
<sname>	basename of custom script ("FOO.cus" in the example)
<tdir>	"**?**"
<tname>	"**?**"

Custom Install Script Predefined Environment Variables

The `Install` utility predefines a number of environment variables, as listed and explained below, for use in custom scripts. These variables are helpful or necessary for writing proper custom install scripts. In order to maximize portability of custom install scripts, use these variables rather than other techniques of obtaining information whenever possible. (For instance, use the value of $CPUTYPE to determine the installation platform, rather than the UNIX **uname** command, which is not available on all platforms).

These environment variables are also available for use in X resource file templates. See the *X Resource File Template Creation* section on page 5-26 for more information.

$AUTO

If not null, $AUTO indicates that `Install` is running in auto (reduced interaction) mode. If null, $AUTO indicates that `Install` is running in custom (full interaction) mode.

When `Install` is running in custom mode, custom install scripts are expected to interact freely with the user—interacting as much as is necessary to allow the user full choice in customizing installation of the extension module. When `Install` is running in auto mode, however, custom install scripts should limit user interaction as much as possible, obtaining only those installation parameters that are absolutely necessary to ensure successful installation of the extension module, and assigning default values to installation parameters wherever safe to do so (see Figure 5-3 for example).

Figure 5-3. Script Example Showing Use of $AUTO Environment Variable

```
# If auto mode, set VAR to default value
if [ "$AUTO" ] ; then
VAR="default_value"
# Else custom mode, so obtain VAR value from user
else
    echo -n "Enter value of VAR: "
    read VAR
fi
```

$AWKUTIL

The $AWKUTIL variable should contain the name of a reliable version of the UNIX awk program on the current platform. Since the native awk utility is buggy on some platforms, use of $AWKUTIL is safer.

For example, use the command:

```
$AWKUTIL '$1 = "file:" { print $5 }' $INDEX
```

instead of:

```
awk '$1 = "file:" { print $5 }' $INDEX
```

$CPUTYPE

The $CPUTYPE variable identifies the installation platform, as defined in Table 3-1 on page 3-6. This $CPUTYPE variable is useful for writing custom scripts that contain nonportable installation functions. In this respect, you should use $CPUTYPE in preference to using the UNIX **uname** command (see Figure 5-4 for example).

Figure 5-4. Script Example Showing Use of $CPUTYPE Environment Variable

```
case "$CPUTYPE" in
intel) echo "Installing on NT" ;;
sun4c) echo "Installing on Solaris" ;;
esac
```

$DEV_NULL

The $DEV_NULL variable identifies the output sink device for the platform. The $DEV_NULL variable evaluates to "**/dev/null**" on UNIX platforms and evaluates to "nul:" on Windows NT platforms.

Make absolutely sure that all literal references to "/dev/null" and "nul:" are replaced with $DEV_NULL in your custom scripts. Failure to do this can result in both subtle and severe problems if the custom script is invoked on the wrong platform. (This is especially important for experienced UNIX shell programmers, who tend to use /dev/null in scripts without thinking.)

$IROOT

The $IROOT variable identifies the installation root directory — that is, the current directory at the time that Install was invoked — as follows:

```
CUS_DOCS=$IROOT/OnlineDocs/helpdoc/helpdoc
```

$LD_LIBRARY_PATH

Some X Windows applications use the $LD_LIBRARY_PATH variable to locate shared libraries. You can use this variable to enable the build of such applications within custom install scripts.

$PATH

The $PATH variable is the Bourne shell command search path. Install sets $PATH to allow access to the following tools:

- Standard installation tools ($IROOT/Install-Tools).
- SpectroSERVER installation tools ($SS_TOOLS_ROOT/SS-Tools).
- SpectroGRAPH installation tools ($SG_TOOLS_ROOT/SG-Tools).
- Standard UNIX commands on the installation host.

$RANLIB

The $RANLIB variable is the name of the UNIX ranlib utility on the current platform. $RANLIB should be used instead of ranlib, because ranlib does not exist on some platforms — and on some others is implemented through the use of the UNIX **ar** command.

For instance, use

```
$RANLIB library.a
```

in place of

```
ranlib library.a
```

$SG_ROOT

The `$SG_ROOT` variable identifies the SpectroGRAPH product root directory. The SpectroGRAPH executable and associated files are installed in the `$SG_ROOT/SG` directory.

$SG_SUP_ROOT

The `$SG_SUP_ROOT` variable identifies the SpectroGRAPH product support file root directory. SpectroGRAPH support data files are installed in the `$SG_SUP_ROOT/SG-Support` directory.

$SG_TOOLS_ROOT

The `$SG_TOOLS_ROOT` variable identifies the SpectroGRAPH product tools root directory. SpectroGRAPH support tools are installed in the `$SG_TOOLS_ROOT/SG-Tools` directory.

$SS_ROOT

The `$SS_ROOT` variable identifies the SpectroSERVER product root directory. The SpectroSERVER executable and associated files are installed in the `$SS_ROOT/SS` directory.

$SS_TOOLS_ROOT

The `$SS_TOOLS_ROOT` variable identifies the SpectroSERVER product tools root directory. SpectroSERVER support tools are installed in the `$SS_TOOLS_ROOT/SS-Tools` directory.

Predefined Functions for Custom Install Script

Prior to SPECTRUM 4.0, custom script support functions were defined in a set of include files, which were included in the custom script through use of the "**##include**" directive (similar to the "**#include**" directive in the C programming language). Support for this directive has been discontinued, and the directive is now treated as a Bourne shell comment. Instead, `Install` now provides a standard set of Bourne shell functions that may be used in any custom script. The following is a list of supported functions for use in custom install scripts.

Prior to SPECTRUM 4.0, `Install` also provided support for a number of undocumented custom script functions. Some of these functions were used in custom scripts produced at Cabletron, and may have been cloned into custom scripts produced by VARs. Any custom script support functions not documented in this chapter must be removed from any custom script shipped on SPECTRUM 5.0 media; otherwise, the custom script may fail or behave improperly at installation time.

`app_def_set`

Usage: `app_def_set <varname> <value>`

Description: The `app_def_set` function is used to set a variable value to be expanded within X resource file templates when installing X resource files. `<varname>` is the name of a variable and must begin with "CS_"; `<value>` is any string. (Refer to the *X Resource File Template Creation* discussion on page 5-26, which gives an example of the use and effect of this function in the `xappl.cus` custom script described in that discussion.)

Retired Custom Install Script Variables and Functions

The variables and functions listed below have been "retired" and are no longer supported in custom install scripts for use with SPECTRUM 5.0.

`backup_app_defs`
`deliver_app_defs`
`update_app_defs`

These functions were formerly made available in custom scripts to install X resource files. The method for installing X resource files has been changed to standardize the procedure across all SPECTRUM applications, and these functions are no longer used. For more information on writing, shipping, and installing X resource files, see the *X Resource File Templates* section on page 5-26.

`get_entries`

The **`get_entries`** function was formerly made available in custom scripts to give extension modules access to information in their own index files during installation. The **`get_entries`** interface is somewhat obscure, and its usefulness is limited; while support has been continued to the present release, this interface most likely will be retired in the near future.

$LOG
$LOGDIR

The `$LOG` and `$LOGDIR` environment variables were formerly made available to custom scripts to allow them to manipulate the installation log file or create their own log files. As of this version of the SEI Toolkit, all output of custom scripts is automatically logged to the installation log file, so that custom scripts no longer need to manage their own logging.

$WINDOWING_SYS_BIN

The `$WINDOWING_SYS_BIN` variable identified the windowing system executables (bin) directory. This directory was formerly used to locate tools related to X Windows applications. Use of X Windows tools and libraries in custom scripts is no longer supported.

$WINDOWING_SYS_LIB

The `$WINDOWING_SYS_LIB` variable identified the windowing system library directory. This directory was formerly used to locate static and shared libraries for linking and running executables using X Windows support functions. Use of X Windows tools and libraries in custom scripts is no longer supported.

$WINDOWING_SYS_LIB_APP

The `$WINDOWING_SYS_LIB_APP` variable identified the windowing system X resource (app-defaults) directory. Formerly `$WINDOWING_SYS_LIB_APP` was used in conjunction with the now-retired `backup_app_defs`, `update_app_defs`, and `deliver_app_defs` functions to install X resource files for SPECTRUM applications. The new method for shipping and installing X resource files for SPECTRUM applications is described in the *X Resource File Templates* discussion on page 5-26. Other usages of `$WINDOWING_SYS_LIB_APP` in custom scripts should be replaced by "`$IROOT/app-defaults`".

$WINDOWING_SYS_TKLIB

The `$WINDOWING_SYS_TKLIB` variable identified the windowing system toolkit library directory. This directory was formerly used to locate static and shared libraries for linking and running executables using X/Motif, openwin, or other X Windows toolkits. Use of X Windows tools and libraries in custom scripts is no longer supported.

X Resource File Templates

Many SPECTRUM extensions modules ship and install graphical applications designed to run within the standard SPECTRUM windowing environment. This is seldom done within Level 1 extension modules, whose graphics are normally handled by SpectroGRAPH. Many Level 2 extension modules are independent of SpectroGRAPH, however, and these modules have their own graphical interfaces.

With such an application, it is often necessary to install an X resource file (also commonly called an app-defaults file), which is used by an X-based windowing system to define default values for the application when the application is started. The following discussion describes how to write and ship X resource files for SPECTRUM applications, and how they are installed.

X Resource File Template Creation

The following paragraphs give an example of writing and shipping an X resource file so that it will install into the proper directory for SPECTRUM X resource files and contain the correct values, which depend on the installation environment and user input during the installation. It is highly recommended that you read this discussion before writing and shipping any X resource files.

The example presented in this discussion provides details on the writing, shipping, and installation of the X resource file for a fictitious application, `xappl`, which is designed to run within the SPECTRUM environment. The `xappl` function is to be shipped with an extension module, XMODULE. When installed, the `xappl` resource file should contain resources for a path to a font directory and a path to SpectroGRAPH image files, and it also should determine whether or not `xappl` icons should blink.

Suppose that the installation were performed in the directory `/usr/Spectrum/INST` on a Solaris host whose native X library directory was `/usr/openwin/lib/X11`, and suppose that the installing user chose to have blinking `xappl` icons. For these choices, that user would want the `xappl` resource file to be installed in the SPECTRUM X resource directory with the following contents:

```
// X resource file for "xappl" application
*fontpath:  /usr/openwin/lib/X11/fonts/misc/fonts.dir
*imagepath:  /usr/Spectrum/INST/SG-Support/CsImage
*iconBlink:  true
```

Suppose the installation were performed by a different user, working in the `c:/win32app/Spectrum` area on a Windows NT host having `c:/win32app/nutcxsrv as a` native X library directory, and that this user did *not* want to have blinking `xappl` icons. In this case, that user would want

the `xappl` resource file to be installed in the SPECTRUM X resource directory with the following, quite-different contents:

```
// X resource file for "xappl" application
*fontpath:  c:/win32app/nutcxsrv/fonts\misc/fonts.dir
*imagepath:  c:/win32app/Spectrum/area/SG-Support/CsImage
*iconBlink:  false
```

In other words, the contents of the installed `xappl` resource file (or any other X resource file) is likely to be different for different installations, and you usually cannot know before installation time exactly what values go into the installed X resource file.

In order to allow the contents of the installed X resource file to vary across installation, the `mkmm` utility lets you ship an X resource file template. The template contains variable values that can be determined and filled in by `Install` at installation time, depending on the installation environment and the user input during the installation.

For this fictitious `xappl` application, you would typically create an X resource file template, `xappl`, in the same directory with `XMODULE.i`, the index file for the XMODULE extension module that contains the `xappl` application. You should name the X resource file template `xappl`, because the installed X resource file will have the same name as the template, and it is customary to give the X resource file the same name as the application that uses it — in this case, `xappl`. (You want the template in the same directory with `XMODULE.i`, because `mkmm` looks for X resource files templates in the same directory as the index files.

The contents of the X resource file template `xappl` are:

```
// X resource file for "junktool" application
*fontpath  ${WINDOWING_SYS_LIB}/fonts/misc/fonts.dir
*imagepath:  ${SG_SUP_ROOT}/SG-Support/CsImage
*iconBlink:  ${CS_XAPPL_ICONBLINK}
```

Note that `xappl` contains the variables `${WINDOWING_SYS_LIB}`, `${SG_ROOT}`, and `${CS_XAPPL_ICONBLINK}`, which look much like Bourne shell environment variables. At installation time, `Install` expands these embedded Bourne shell environment variables to their exported values to create the final X resource file.

In X resource file templates, only environment variables of the form `${VAR}` are expanded, where `VAR` is a valid Bourne shell environment variable name starting with an upper or lower case alphabetic character (*not* underscore). The braces *must* be present in this case. Any text not of this form remains unchanged in the expanded X resource file, including variables of the form `$VAR` and also such special Bourne shell variables as `$$`, `$!`, `$0`, `$1`, `$?`, and so on. The `${DS}` variable expands to a dollar sign in X resource file templates,

and this variable can be used to escape variable expansion by using such combinations as `${DS}{VAR}`, which expands to `${VAR}`).

In `xappl`, the variables `${WINDOWING_SYS_LIB}` and `${SG_SUP_ROOT}` are predefined by `Install` to expand to the X library directory and the SpectroGRAPH support file root directory, respectively — precisely the values we want in the installed X resource file at those points. (Refer to the *Custom Install Script Predefined Environment Variables* discussion on page 5-20.)

The `${CS_XAPPL_ICONBLINK}` variable, whose value is "true" or "false" according to whether the user wants blinking `appl` icons, is not predefined by `Install`, but is a user-created variable. Its value must be obtained and communicated to the X resource file template via a custom script. The custom script asks if the user wants blinking `appl` icons, obtains the answer, and uses the `app_def_set` function to set the appropriate value of `${CS_XAPPL_ICONBLINK}` for expansion within the X resource file.

To accomplish this, for the given hypothetical programming situation, you would create a custom script, named `xappl.cus`, as shown in Figure 5-5, in the same directory with `xappl` and `XMODULE.i`.

Figure 5-5. Sample Custom Script for Installation of xappl Application

```
 1  ################################################################
 2  # SG custom script for installation of "xappl" application
 3  ################################################################
 4
 5  # If Install is running in auto mode, you are not allowed
 6  # to interact with the user.
 7  if [ "$AUTO" ] ; then
 8
 9      # Assume user does not want blinking icons
10      BLINK=false
11
12  # Else Install is running in custom mode, so you can
13  # interact with the user.
14  else
15
16      # Ask if the user wants blinking icons
17      echo "Do you want blinking xappl icons?"
18      read ANS
19
20      # To blink or not to blink?
21      case $ANS in
22      [YyTt]*)  BLINK=true ;;
23      *)        BLINK=false ;;
24      esac
25  fi
26
27  # Set value of ${CS_XAPPL_DOBLINK} for later expansion
28  # within xappl's X resource file.
29  app_def_set CS_XAPPL_DOBLINK $BLINK
30  exit 0
```

Finally, you also would add entries to `XMODULE.i` to ship the X resource template file, `xappl`, and the custom install script, `xappl.cus` (assuming that `XMODULE.i` already contains an entry for shipping the `xappl` application itself):

```
file:   SG appdef ? xappl ? ?
file:   SG cus ? xappl.cus ? ?
```

At installation time, `Install` will invoke the custom script, `xappl.cus`. If `Install` is running in auto mode, `xappl.cus` assumes the user does not want blinking icons and sets `$BLINK` to "`false`." If `Install` is running in custom mode, `xappl.cus` asks if the user wants blinking `xappl` icons, and sets `$BLINK` to "`true`" or "`false`" accordingly. Finally, `xappl.cus` uses `app_def_set` to set the value of `${CS_XAPPL_DOBLINK}` to the value of `$BLINK` (either "true" or "false") for expansion within the custom script.

Note that all custom scripts share the same variable name space with respect to `app_def_set`. If two custom scripts use `app_def_set` to set the same variable (`$CS_JOE`, for `example`), then only one of the values is expanded in all shipped X resource template files. For this reason, it is a good idea to name your environment variables in such a way as to reduce the likelihood of their use in other custom scripts. `${CS_XAPPL_ICONBLINK}` was used in the preceding example, for instance, because other custom scripts are not likely to use `XAPPL` as part of their variable names.

SPECTRUM Graphical Application Standards

When writing graphical applications to run within the SPECTRUM environment, you should adhere to the following standards for SPECTRUM graphical applications:

- The application should be based on X11R5 or later version of X.
- X resource files for the application should be installed using the methods outlined in the *X Resource File Template Creation* section on page 5-26.
- Before running the application, source in the standard environment setup file `Install-Tools/setup.ksh` (for Bourne shell or Korn shell) or `Install-Tools/setup.csh` (for C shell) shipped with every SPECTRUM distribution. You should include the text of these files in your `.profile` or `.cshrc` file (or equivalent) as appropriate, if you intend to use SPECTRUM graphical applications on a regular basis.

VAR-developed applications derive the following benefits from adhering to this standard:

- The application will work in conjunction with other SPECTRUM applications.

- The application will be more easily portable across supported SPECTRUM platforms.

stdmenu and spmamap files

`Install` uses `stdmenu` files (shipped via the entry file: `SG stdmenu`) to build the CsStdMenu menu configuration file used by SpectroGRAPH. `Install` uses `spmamap` files (shipped via the `file: SG spmamap` entry) to build the file `CsSPMAMAP`, used by the SPECTRUM generic application launcher.

Prior to building `CsStdMenu` and `CsSPMAMAP`, the `spmamap` and `stdmenu` files undergo the same environment variable expansion described above for X resource file templates.

PIB Contribution Files

`Install` uses PIB contribution files to build the SpectroGRAPH PIB database. When installed, the PIB database resides in the `SG-Support/CsPib` directory (relative to the installation root directory) and consists of the View Control file (`CsViewControl`) and a number of PIB files with the "`.pib`" extension. (For more information on the purpose of the files in the PIB database, refer to the ***SPECTRUM IIB Editor Guide***.)

During installation, extension modules may want to add entries to any of the various files in the PIB database. Prior to SPECTRUM 4.0, this was done by including `pib:` entries in the extension module's index file.

The use of `pib:` index file entries is fairly straightforward. For example, if the DCNET extension needed to add the line

```
CtTRingApp/CtTRApp.DIBM,CtTRingApp
```

to the `SG-Support/CsPib/CsTRDev.pib` PIB file at installation time, the developer would have accomplished this by adding the `pib:` entry

```
pib: CsPib/CsTRDev.pib \
     CtTRingApp/CtTRApp.DIBM,CtTRingApp
```

to the index file DCNET.i.

The mechanism for adding information to the PIB database was changed effective with SPECTRUM 4.0. Instead of using index file `pib:` entries to update the PIB database during installation, it is necessary to create a PIB contribution file, which describes the PIB database updates. The following subsection outlines how to create a PIB contribution file. The new PIB

contribution file should contain one entry for each `pib:` entry in the old index file.

Manual Generation of PIB Contribution Files

If you are upgrading an older index file into a SPECTRUM 5.0 index file, you should use `mkmmconv` as your tool of choice. If there are **`pib:`** entries in your index file, `mkmmconv` will automatically generate a PIB contribution file for you. The complete process of using `mkmmconv` to upgrade an index file is described in *Index File Conversion*, starting on page 3-37.

If you are upgrading an older index file by hand, the process is still fairly simple. First, create a PIB contribution file in the same directory that contains the index file. The PIB contribution file should have the same name as the extension module index file, but with a `.p` extension. If the index file is `DCNET.i`, for example, the PIB contribution should be `DCNET.p`.

For each `pib:` entry in `DCNET.i`, add a converted entry to the PIB contribution file. The entry is converted by removing the `pib:` label and the `CsPib/` level from the PIB file name. For example, if `DCNET.i` contains the following **`pib:`** entry:

```
   pib: CsPib/CsTRDev.pib
CtTRingApp/CtTRApp.DIBM,CtTRingApp
```

then add the following converted entry:

```
   CsTRDev.pib CtTRingApp/CtTRApp.DIBM,CtTRingApp
```

to `DCNET.p`.

Once all of the `pib:` entries have been transferred from the index file to the PIB contribution file, remove all of the `pib:` entries from the index file. Then add an entry to ship the PIB contribution file. For the given example, you would add the following entry to `DCNET.i`:

```
   file: SG pib ? DCNET.p ? ?
```

Inclusion of this entry ensures that the PIB contribution file gets shipped and processed during installation of the extension module.

Finally, add both the modified index file (in this case, `DCNET.i`) and the new PIB contribution file to your source control area.

If you are developing a new extension module that needs to modify the PIB database during installation, the process is similar. Create a new PIB contribution file as described above, add the necessary entries to the PIB

contribution file, and add the `file:` entry to the index file as described above. Again, you must also add the modified index file and the new PIB contribution file to your source control area.

Chapter 6

Use Scenarios

This chapter illustrates two typical scenarios of SEI Toolkit use, showing its flexibility in the type of components that you can install, configure, and operate with SPECTRUM. The first scenario demonstrates the process of accomplishing MM vtape distribution. The second scenario illustrates how you can integrate other SPECTRUM Level 2 Tools.

Scenario 1: MM Vtape Distribution

This scenario describes how to use the SEI Toolkit to generate a distribution vtape for a management module (MM). You can then use the newly created distribution vtape to install the management module at a customer's site. This scenario uses SM-MYR1001 as the name of a fictitious management module. In addition, this scenario presumes that the SEI Toolkit is installed in the `/usr/manmods/devarea` directory.

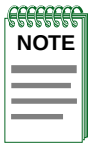
To generate an MM distribution vtape for MM SM-MYR1001, you must perform the following functions:

1. Install the SEI Toolkit.
2. Set up a development area.
3. Create an index file.
4. Create an MM archive file.
5. Create a manufacturing area.
6. Add the management module to the manufacturing area.
7. Distribute the management module to vtape.

The following sections provide details about each of these functions, in sequence.

Installing the SEI Toolkit

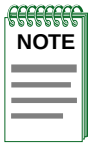
The SEI Toolkit contains all files and utilities necessary to create index files, to create and set up manufacturing areas, to create parts files, and to distribute VAR-developed management modules to vtape, from which they can be integrated into SPECTRUM at the customer's site. The SEI Toolkit is platform dependent—that is, you must purchase a separate SEI Toolkit for each platform (Solaris or NT) for which you are developing management modules.



All SEI utilities must reside in their original installed location within the SEI Toolkit.

To install the SEI Toolkit, use the standard SPECTRUM install script**,** `Install`. The `Install` script retrieves the SEI Toolkit files from the distribution medium and then calls the custom install script `insdk.cus` to build the correct directory structure and to install the toolkit files. For more detailed information about `Install`, refer to the ***SPECTRUM Installation Guide***.

Setting Up a Development Area



The development area coding outlined in this section is a fictitious example for this scenario, and is not to be taken literally as a guide for the development of real Level 1 management modules. The coding shown here is meant only as an example of how to distribute files from a development area to a medium.

The SEI Toolkit is installed in the directory `/usr/manmods/devarea`.

Set and export the environment variable `$DEVAREA` to `/usr/manmods/devarea`.

```
DEVAREA=/usr/manmods/devarea
export DEVAREA
```

Figure 6-1 identifies the files that comprise the development area for this scenario. Those files preceded by an asterisk are files that are to be distributed in the `SM-MYR1001` management module. The remaining files are source files, which are not to be distributed in the `SM-MYR1001` management module.

Figure 6-1. Contents of /usr/manmods/devarea Development Area for Scenario 1

```
  /usr/manmods/devarea/myrtr/Makefile
* /usr/manmods/devarea/myrtr/SG-Support/CsEvFormat/Event003e0000
* /usr/manmods/devarea/myrtr/SG-Support/CsEvFormat/Event003e0002
* /usr/manmods/devarea/myrtr/SG-Support/CsGib/mrDecNetApp/CsCrcuitGTb.30
* /usr/manmods/devarea/myrtr/SG-Support/CsGib/mrDecNetApp/CsRouteGrp.30
* /usr/manmods/devarea/myrtr/SG-Support/CsGib/mr_SerIF_Port/CsStandard.30
* /usr/manmods/devarea/myrtr/SG-Support/CsGib/my_router/CsIfConfig.30
* /usr/manmods/devarea/myrtr/SG-Support/CsGib/my_router/CsIfDMap.30
* /usr/manmods/devarea/myrtr/SG-Support/CsIib/mrIPApp/mrIPApp.AIBase
* /usr/manmods/devarea/myrtr/SG-Support/CsIib/my_router/Banner.Ifc
* /usr/manmods/devarea/myrtr/SG-Support/CsIib/my_router/NWelRtr.ISd
* /usr/manmods/devarea/myrtr/SG-Support/CsImage/Makefile
* /usr/manmods/devarea/myrtr/SG-Support/CsImage/myRouter.csi
* /usr/manmods/devarea/myrtr/SG-Support/CsRib/my_router/my_Router.rib
* /usr/manmods/devarea/myrtr/SG-Support/CsScript/start_sm
* /usr/manmods/devarea/myrtr/SG-Support/CsVendor/my_Rtr/my_router/AlertMap
* /usr/manmods/devarea/myrtr/SG-Support/CsVendor/my_Rtr/my_router/EventDisp
  /usr/manmods/devarea/myrtr/SG-Support/Makefile
  /usr/manmods/devarea/myrtr/db/Makefile
* /usr/manmods/devarea/myrtr/db/SMMYR.e
  /usr/manmods/devarea/myrtr/db/SMMYR.m
  /usr/manmods/devarea/myrtr/db/SMMYR.v
  /usr/manmods/devarea/myrtr/db/SMMYR1p.m
  /usr/manmods/devarea/myrtr/icon/CsIGLIntAct.cc
* /usr/manmods/devarea/myrtr/icon/CsIGLIntAct.o
  /usr/manmods/devarea/myrtr/icon/CsIGMTxtAct.cc
* /usr/manmods/devarea/myrtr/icon/CsIGMTxtAct.o
  /usr/manmods/devarea/myrtr/icon/Makefile
  /usr/manmods/devarea/myrtr/include/CsIGLIntAct.h
  /usr/manmods/devarea/myrtr/include/CsIGMTxtAct.h
  /usr/manmods/devarea/myrtr/install/Makefile
* /usr/manmods/devarea/myrtr/install/SM-MYR1001.cus
* /usr/manmods/devarea/myrtr/install/SM-MYR1001.i
  /usr/manmods/devarea/myrtr/intel/CsDNAppMI.cc
* /usr/manmods/devarea/myrtr/intel/CsDNAppMI.o
  /usr/manmods/devarea/myrtr/intel/CsIHInRdExt.cc
* /usr/manmods/devarea/myrtr/intel/CsIHInRdExt.o
  /usr/manmods/devarea/myrtr/intel/Makefile
  /usr/manmods/devarea/myrtr/mibs/mrTFilter.mib
  /usr/manmods/devarea/myrtr/mibs/mrVines.mib
```

Creating an Index File

1. Use a text editor, such as ***vi***, to create the index file, `SM-MYR1001.`***i***.
2. Initialize the index file by defining the extension module name, vendor name, irev version number, version number, and level, as shown below. For specific information about index file formats, refer to *Chapter 3, Index Files*.

```
mm:SM-MYR1001
vend:Cabletron
irev:01.00.01.001
rev:2.0BETA0
level:1
```

3. Add a database export file (`*.e` files), containing the appropriate model types and model type list files (`*.m` files). You export the model types (relations, rules, attributes, etc.) from the SPECTRUM database with `dbtool`, a command line version of the Model Type Editor. The model type list files, also called description files, are text files that contain a list of all model type handles required by the extension module. (For more information about these `dbtool` output files, refer to Chapter 4 of the ***Model Type Editor User's Guide***.)

   The contents of the `SMMYR.m` file for this scenario are:

```
0x000103a4
0x000103a5
0x000103a6
```

   The model-type handles come from the SPECTRUM database; you should keep a record of these handles when creating them through the Model Type Editor. After `dbtool` has completed processing, there will be a new binary file, `SMMYR.e`. To ship this file in the index file, add the following entry:

```
file: SS db ${MYRTR_DEV}/db SMMYR.e ? ?
```

   Note that, when processing this **`file:`** entry, `mkmm` will try to find `SMMYR.e` in the `${MYRTR_DEV}/db` directory, which must expand to `/usr/manmods/devarea/myrtr/db`. To provide for this situation, set `${MYRTR_DEV}` to `/usr/manmods/devarea/myrtr` and export its value before invoking `mkmm` to build the `SM-MYR1001.tar` management module archive. (See *Creating an MM Archive File*, on page 6-6.)

4. Add the SG-Support files (for example, iibs, gibs, images) needed for this extension module. The SG-Support files listed in the `SM-MYR1001` extension module index file are as follows:

```
deflt: SG gib sdir $(MYRTR_DEV)/SG-Support
deflt: SG iib sdir $(MYRTR_DEV)/SG-Support
deflt: SG image sdir $(MYRTR_DEV)/SG-Support
deflt: SG other sdir $(MYRTR_DEV)/SG-Support
deflt: SG rib sdir $(MYRTR_DEV)/SG-Support
deflt: SS vfile $(MYRTR_DEV)/SG-Support
file: SG gib ? CsGib/mrDecNetApp/CsCrcuitGTb.30 ? ?
file: SG gib ? CsGib/mrDecNetApp/CsRouteGrp.30 ? ?
file: SG gib ? CsGib/mr_SerIF_Port/CsStandard.30 ? ?
file: SG gib ? CsGib/my_router/CsIfConfig.30 ? ?
file: SG gib ? CsGib/my_router/CsIfDMap.30 ? ?
file: SG iib ? CsIib/mrIPApp/mrIPApp.AIBase ? ?
file: SG iib ? CsIib/my_router/Banner.Ifc ? ?
file: SG iib ? CsIib/my_router/NWelRtr.ISd ? ?
file: SG image ? CsImage/myRouter.csi ? ?
file: SG other ? CsEvFormat/Event003e0000 ? ?
file: SG other ? CsEvFormat/Event003e0002 ? ?
file: SG other ? CsScript/start_sm ? ?
file: SG rib ? CsRib/my_router/my_Router.rib ? ?
file: SG icon ${MYRTR_DEV}/icon CsIGLIntAct.o ? ?
file: SG icon ${MYRTR_DEV}/icon CsIGMTxtAct.o ? ?
file: SS vfile ? CsVendor/my_Rtr/my_router/AlertMap ? ?
file: SS vfile ? CsVendor/my_Rtr/my_router/EventDisp ? ?
```

5. Add the references to new intelligence added to this extension module. New intelligence consists of Inference Handlers (ih) and Model Intelligence (mi) nodes. An Inference Handler is the C++ coding that causes the SpectroSERVER to manipulate or act on a specific piece of intelligence about a device. The Model Intelligence node gathers the model-specified Inference Handler into the SpectroSERVER. The ih and mi entries from the `SM-MYR1001` extension module are:

```
file: SS ih ${MYRTR_DEV}/intel CsIHInRdExt.o ? ?
file: SS mi ${MYRTR_DEV}/intel CsDNAppMI.o ? ?
```

6. The `SM-MYR1001` extension module uses a custom install script to perform special functions that the SPECTRUM Install script (`Install`) does not provide. The labels used for custom install scripts are `file: SS cus …` and `file: SG cus ….` The custom install script files may also need other files to successfully install the extension module. The labels that include these files are `file: SS xtn …` and `file: SG xtn ….` For more information about the labels for custom install scripts and external files, refer to *Index File Syntax Rules*, on page 3-1.

   The custom install script entry for the `SM-MYR1001` extension module is:

```
file: SG cus ? SM-MYR1001.cus ? ?
```

Figure 6-2 shows the index file for the extension module SM-MYR1001.

Figure 6-2. Index File SM-MYR1001.i for Extension Module SM-MYR1001

```
mm:  SM-MYR1001
vend:  Cabletron
irev:  01.00.01.001
rev:  2.0BETA0
level:  1

head:  Install mmmdesc ? SM-M{R-100}.mmd ? ?
deflt: SG gib sdir $(MYRTR_DEV)/SG-Support
deflt: SG iib sdir $(MYRTR_DEV)/SG-Support
deflt: SG image sdir $(MYRTR_DEV)/SG-Support
deflt: SG other sdir $(MYRTR_DEV)/SG-Support
deflt: SG rib sdir $(MYRTR_DEV)/SG-Support
deflt: SS vfile $(MYRTR_DEV)/SG-Support
file: SG gib ? CsGib/mrDecNetApp/CsCrcuitGTb.30 ? ?
file: SG gib ? CsGib/mrDecNetApp/CsRouteGrp.30 ? ?
file: SG gib ? CsGib/mr_SerIF_Port/CsStandard.30 ? ?
file: SG gib ? CsGib/my_router/CsIfConfig.30 ? ?
file: SG gib ? CsGib/my_router/CsIfDMap.30 ? ?
file: SG iib ? CsIib/mrIPApp/mrIPApp.AIBase ? ?
file: SG iib ? CsIib/my_router/Banner.Ifc ? ?
file: SG iib ? CsIib/my_router/NWelRtr.ISd ? ?
file: SG image ? CsImage/myRouter.csi ? ?
file: SG other ? CsEvFormat/Event003e0000 ? ?
file: SG other ? CsEvFormat/Event003e0002 ? ?
file: SG other ? CsScript/start_sm ? ?
file: SG rib ? CsRib/my_router/my_Router.rib ? ?
file: SG icon ${MYRTR_DEV}/icon CsIGLIntAct.o ? ?
file: SG icon ${MYRTR_DEV}/icon CsIGMTxtAct.o ? ?
file: SS vfile ? CsVendor/my_Rtr/my_router/AlertMap ? ?
file: SS vfile ? CsVendor/my_Rtr/my_router/EventDisp ? ?
file: SS ih ${MYRTR_DEV}/intel CsIHInRdExt.o ? ?
file: SS mi ${MYRTR_DEV}/intel CsDNAppMI.o ? ?
file: SG cus ? SM-MYR1001.cus ? ?
```

Creating an MM Archive File

Use the mkmm tool to create the archive file SM-MYR1001.tar. The mkmm tool reads the index file for the management module and builds a single extension module archive file SM-MYR1001.tar. Packaging the management module into a single archive file prepares it to be added to a manufacturing area for shipment on installable media. Before running mkmm, set and export all environment variables referenced in the index file, which in this case is the single "${MYRTR_DEV}" environment variable. Then call mkmm to build the

archive file. For detailed information about `mkmm`, refer to *Development: mkmm*, on page 4-7.

```
MYRTR_DEV=${DEVAREA}/myrtr
export MYRTR_DEV
${DEVAREA}/INSDK/mkmm SM-MYR1001.i
```

Creating the Manufacturing Area

Use the `mkmanf` tool to build a manufacturing area. Set the environment variable `$MANF` to `/usr/manmods/devarea/manfarea`. Use the UNIX **`rmdir`** command to remove any existing `$MANF` directory and then use the related ***`mkdir`*** command to create a new, empty `$MANF` manufacturing area. After exporting the `$MANF` variable in order to set up the environment variable, call `mkmanf` to build the manufacturing area. Note that you can also use `mkmanf` to compress or to uncompress the management module records in an existing manufacturing area, thereby saving a significant amount of space in the manufacturing area as well as on any distribution medium generated from the manufacturing area. For detailed information about `mkmanf`, refer to *Manufacturing: mkmanf*, on page 4-15.

```
MANF=/usr/manmods/devarea/manfarea
/bin/rm -rf $MANF
mkdir $MANF
export MANF
$INSDK/mkmanf
```

Adding the MM Archive File to the Manufacturing Area

1. Move the `SM-MYR1001.tar` file (which was created by `mkmm`) to the manufacturing area and then make that area the current directory, as follows:

   ```
   cp $DEVAREA/myrtr/install/SM-MYR1001.tar $MANF
   cd $MANF
   ```

2. Use the `mkarea` tool to integrate `SM-MYR1001.tar` into the manufacturing area. The `mkarea` tool extracts all the components of the management module from the management module archive file and moves them to their respective directories within the manufacturing area. Call `mkarea` with the “parts” argument to build a “parts” file for `mkdist`. With “plist” specified, `mkarea` puts the names of all management modules

in the manufacturing area into a file named `partlist`. For detailed information about `mkarea`, refer to *Manufacturing: mkarea*, on page 4-20.

```
$INSDK/mkarea parts
```

or

```
$INSDK/mkarea plist=partlist
```

Distributing the management module to Vtape

Use the `mkdist` tool to build a distribution to virtual tape. Call `mkdist`, with "**`plist=partlist`**" as an argument, to build a distribution with all of the extension modules listed in the "`parts`" file.

```
cd $MANF
$INSDK/mkdist plist=partlist  vtape=MANF/vtap
```

or

```
cd $MANF
$INSDK/mkdist plist=partlist  tape=dev/nvst8
```

Cabletron and larger customer users would now produce a CD-ROM or use direct download techniques to convey the just completed distribution copy to a customer site, using the `Install` program to integrate this copy of `SM-MYR1001` into SPECTRUM at that site. (Refer to the ***SPECTRUM Installation Guide*** for instructions.)

Using a CD-ROM approach for vtape distribution would be pretty much the same as described for NT distribution in *CD-ROM Virtual Tape (CD-ROM Vtape)*, on page 4-30.

Third-party developers who do not have CD-ROM capability can ship the completed vtape on magnetic tape, as discussed below.

Presume that you have used the `mkdist` to create the vtape, as follows:

```
mkdist vtape=/user/development/vtape
```

Mount a system-compatible tape in your tape drive and then perform whatever setup procedures are required to get the tape in a properly rewound condition, ready for recording. Then change to the directory in which you created the vtape and tar the vtape file onto the magnetic tape:

```
cd /user/development
tar cvf /dev/rmt/0m vtape
```

Extraction/installation at the customer site uses the opposite approach. You first mount the magnetic tape in the system-connected tape drive and perform whatever setup procedures are required to get the tape in a properly rewound condition, ready to be read into the system. Then extract the vtape files from the magnetic tape, as follows:

```
cd /user/vtapes
tar xvf /dev/rmt/0m
```

Run the `Install` program, as follows, and give the extracted vtape as the vtape location in the install gui (in this example, `/user/vtapes/vtape`) when that information is solicited.

```
cd /usr/Spectrum
./Install
```

Scenario 2: Developer's Tools Integration

This scenario shows how to use the SEI Toolkit to integrate several of the SPECTRUM Level 2 Developer's Tools. The distribution generated in this scenario consists of the following toolkits:

- SPECTRUM External Protocol Interface Toolkit (EPAPI)
- SPECTRUM Inference Handler Toolkit (IHAPI)
- SPECTRUM SpectroSERVER API Toolkit (SSAPI)

The scenario assumes that these toolkits were purchased and have been installed in the area used to install SPECTRUM.

1. Create an area called `devarea` for the development of the manufacturing area for the distribution:

   ```
   mkdir devarea
   ```

2. Extract the toolkit index files and their corresponding custom install scripts from the Install record of the distribution.

   ```
   cd devarea
   tar xv epapi.i epapi.cus ihapi.i ihapi.cus ssapi.i \
   ssapi.cus comm.i comm.cus globl.i globl.cus vpapi.i \
   vpapi.cus vwapi.i vwapi.cus
   ```

The source code used in this scenario comes from the toolkits installed earlier. Typically, the source files for customer- or VAR-developed extensions reside in the development area.

3. Initialize the environment variables that are used by the Installation Toolkit and the index files for the toolkits. The easiest way is to create a simple ASCII file and source it. Figure 6-3 shows a representative `init_env` file for this scenario, using the Bourne shell (`sh`).

Figure 6-3. Representative init_env file for Scenario 2 (Using Bourne shell)

```
#!/bin/sh
    # Initialize the environment variables used by
    # the Installation Toolkit tools.
MANF="/usr/SPECTRUM/devarea/manfarea"
    # Initialize the environment variables used by
    # the EPAPI toolkit index file.
EPTOP="/usr/SPECTRUM"
EPAPI="EPAPI"
    # Initialize the environment variables used by
    # the IHAPI toolkit index file.
IHTOP="/usr/SPECTRUM"
IHAPI="IHAPI"
    # Initialize the environment variables used by
    # the SSAPI toolkit index file.
SSTOP="/usr/SPECTRUM"
SSAPI="SSAPI"
    # Initialize the environment variables used by
    # the other toolkit index files that are included
    # with the IHAPI and SSAPI toolkits.
COMMTOP="/usr/SPECTRUM"
COMM="COMM"
GLOBLTOP="/usr/SPECTRUM"
GLOBL="GLOBL"
VPTOP="/usr/SPECTRUM"
VPAPI="VPAPI"
VWTOP="/usr/SPECTRUM"
VWAPI="VWAPI"
    # Export all environment variables.
export MANF EPTOP EPAPI IHTOP IHAPI SSTOP SSAPI \
    COMMTOP COMM GLOBLTOP GLOBL VPTOP VPAPI VWTOP VWAPI
```

The `init_env` file now initializes and exports all of the environment variables that are needed by the SEI tools and the other Toolkit index files. The SPECTRUM installation area is `/usr/SPECTRUM`. The environment variables can be inherited into the environment by typing:

```
. init_env
```

4. Use the `mkmm` tool to build the tar files for the toolkits:

```
/usr/SPECTRUM/INSDK/mkmm epapi.i ihapi.i ssapi.i \
comm.i globl.i vpapi.i vwapi.i
```

During the process of trying to build a tar file, `mkmm` may detect one or more problems. If a problem occurs, `mkmm` will display an error message to report that event. Suppose, for example, that the following error message is displayed while building the EPAPI Toolkit's tar file, `epapi.tar`:

```
*** ERRORS have occurred, please refer to epapi.err! ***
```

Viewing the contents of `epapi.err` may show that `mkmm` could not find the file `/usr/SPECTRUM/EPAPI/demo/Makefile.demo` (so that the UNIX tar command detected an error while building the `epapi.tar` file).

```
more epapi.err
epapi.err
#########
/usr/SPECTRUM/EPAPI/demo/Makefile.demo not found
tar: EPAPI/demo/Makefile.demo: No such file or \
directory
```

To resolve this problem, copy `Makefile` to `Makefile.demo` and then rerun `mkmm` to create a new EPAPI tar file `epapi.tar`.

```
cd /usr/SPECTRUM/EPAPI/demo
cp Makefile Makefile.demo
cd /usr/SPECTRUM/devarea/
/usr/SPECTRUM/INSDK/mkmm epapi.i
```

5. Now that all of the toolkits have been successfully built, you can create a manufacturing area. Since the environment variable `$MANF` was set to `/usr/SPECTRUM/devarea/manfarea` in Step 3, create the "`${MANF}`" manufacturing area directory, and then execute the SEI `mkmanf` tool.

```
mkdir ${MANF}
/usr/SPECTRUM/INSDK/mkmanf
```

6. After `mkmanf` has completed, move all of the toolkit tar files to the `${MANF}` manufacturing area directory and then change to that directory.

```
mv *.tar ${MANF}
cd ${MANF}
```

7. Use the SEI `mkarea` tool to integrate the toolkit's tar files into the manufacturing area. Include the "**`parts`**" option to build a "`parts`" file, which is used when building the distribution.

```
/usr/SPECTRUM/INSDK/mkarea plist=partlist
```

8. After `mkarea` has completed, rename the "`parts`" file with a more descriptive filename. This ensures that the file is not overwritten the next time you or someone else uses `mkarea` in this manufacturing area.

```
mv partlist devarea_parts
```

9. Using the "vtape" argument, build a distribution with the SEI `mkdist` tool.

   To build a distribution with all of the toolkits listed in the "parts" file, include the "`parts`" file's new filename as an argument, as shown:

```
/usr/SPECTRUM/INSDK/mkdist devarea_parts
```

10. You can now install and integrate the toolkits on the distribution into another SPECTRUM site, using `Install` (refer to the ***SPECTRUM Installation Guide***).

Appendix A

Errors and Warnings

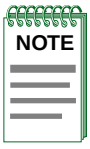
*This appendix contains descriptions of error messages (including warnings and informational messages) generated by **mkmmconv**, **mkmmdeps**, and **mkmm** tools in the SEI Toolkit.*

Output Messages

For SPECTRUM 4.0, much work had been done to standardize the output messages from the SEI Toolkit tools, and to enhance their consistency, completeness, and usefulness. Unfortunately, not all of the SEI Toolkit tools have yet undergone output standardization. The tools that have standardized output at this time are the `mkmmconv`, `mkmmdeps`, and `mkmm` tools.

In the standardized tools, output messages are grouped into three categories:

- **Errors**. When a tool encounters a problem that it cannot correct without user intervention, it outputs an error message to standard output. Error messages always start with an "`Error:`" label. The output of an error message implies that the tool will fail. The tool user should address the reported problem and then rerun the tool.
- **Warnings**. When a tool encounters a problem that it can correct without user intervention, it outputs a warning message and corrects the problem. Warning messages always start with a "`Warning:`" label. The output of a warning message does not imply that the tool will fail. The tool user should address the problem when convenient.



Some of the warning messages produced by the SPECTRUM 4.0 version of SEI Toolkit tools will become errors in future versions of the SEI Toolkit. This means that problems that now merely generate warnings will in the future cause tools to fail. To avoid future problems, therefore, you should address both *errors* and *warnings.*

- **Informative messages**. In addition to errors and warnings, tools may print informative messages that give the user information or help the user

monitor the progress of the tool. Informative messages do not have a label and do not require user action.

mkmmconv Errors

The following is a list of errors and warnings that may be generated by the mkmmconv tool. Any such errors or warnings will be displayed to the screen and also will be saved in files generated as <mmname>.err or <mmname>.warn, respectively, for subsequent viewing. The text following each individual output explains what causes the given message — and, when appropriate, outlines what you should do in response to that particular message.

```
Created PIB contribution file <mmname>.p.
```

Problem: This message does not indicate an error. It simply reports that the mkmmconv tool has found **pib:** entries in the index file and has successfully created a PIB contribution file called <mmname>.p.

Resolution: If you are upgrading an existing index file for compatibility with SPECTRUM 5.0, add <mmname>.p to your source control area, as described in the section *Index File Conversion*, starting on page 3-37.

```
Created converted index file <mmname>.i.conv.
```

Problem: This message does not indicate an error. The mkmmconv tool has created the converted index file identified as <mmname>.i.conv.

Resolution: If you are upgrading an existing index file for compatibility with SPECTRUM 5.0, replace the contents of <mmname>.i with the contents of <mmname>.i.conv in your source control area, as described in *Index File Conversion*, starting on page 3-37.

```
Error: Cannot open error file <mmname>.err.
```

Problem: The mkmmconv tool detected errors during the index file conversion, but was unable to open <mmname>.err to save the associated error messages via a write operation. Note that this error message, itself, also cannot be written to the <mmname>.err error file, which cannot be opened.

Resolution: This error message usually indicates that you do not have privileges to create files within the directory in which mkmmconv was invoked (for instance, you may be on a read-only file system). It could also indicate serious disk space or system problems. Resolve the problem and then rerun mkmmconv.

```
Error: File <mmname>.i: \
Cannot open PIB contribution file <mmname>.p.
```

Problem: The mkmmconv tool detected **pib:** entries in the index file but was unable to open the PIB contribution file, <mmname>.p, to store them via a write operation.

Resolution: This er message usually indicates that you do not have privileges to create files within the directory in which mkmmconv was invoked (for instance, you may be on a read-only file system). It could also indicate serious disk space or system problems. Resolve the problem and then rerun mkmmconv.

```
Error: File <mmname>.i: \
Cannot open converted index file <mmname>.i.conv.
```

Problem: The mkmmconv tool was able to successfully convert <mmname>.i but was unable to open <mmname>.i.conv to store the converted index file via a write operation.

Resolution: This error message usually indicates that you do not have privileges to create files within the directory in which mkmmconv was invoked (for instance, you may be on a read-only file system). It could also indicate serious disk space or system problems. Resolve the problem and then rerun mkmmconv.

```
Error: File <mmname>.i: No valid index file entries.
```

Problem: The mkmmconv tool was unable to find any valid index file entries in <mmname>.i.

Resolution: <mmname>.i is probably an empty file or else is not in index file format. Correct the condition and then rerun mkmmconv.

```
Error: Line <num> of <mmname>.i: \
"<label>" entry has fewer than <ct> fields.
```

Problem: The mkmmconv tool was unable to convert an entry because the entry had too few fields and therefore did not contain enough information to perform the conversion.

Resolution: Fix the offending entry manually and then rerun mkmmconv.

Prior to SPECTRUM 4.0, although index file entries technically did not allow extra fields to the right, the mkmm *tool often silently ignored extra fields in many entries.* mkmmconv *will not ignore these extra fields, however, and will reject such entries as having too many fields.*

```
Error: Line <num> of <mmname>.i: \
"<label>" entry has more than <ct> fields.
```

Problem: The mkmmconv tool was unable to convert an entry because the entry had too many fields. Converting such an entry would require throwing away the information contained in the extra fields.

Resolution: Fix the offending entry manually and then rerun mkmmconv.

```
Error: Line <num> of <mmname>.i: Illegal PIB file <filename>.
```

Problem: The mkmmconv tool detected a **pib:** entry with an illegal PIB file field. Any PIB file not in the CsPib directory is illegal; in addition, CsPib/CsVnmDef and CsPib/CsAlarm.pib are illegal.

Resolution: Remove **pib:** entries with PIB file fields CsPib/CsVnmDef or CsPib/CsAlarm.pib from <mmname>.i manually and either fix or remove any other **pib:** entries with invalid PIB files. Then rerun mkmmconv.

```
Error: Line <num> of <mmname>.i: \
Illegal entry label "<label>".
```

Problem: The mkmmconv tool has detected an entry that has an invalid label, which it cannot convert. The label may be misspelled, or it may be one of a few earlier-version index file labels (for example, **gibdir:**) that are no longer supported.

Resolution: Fix or remove the offending <mmname>.i entry, as appropriate, and then rerun mkmmconv.

```
Error: Line <num> of <mmname>.i: Non-text character.
```

Problem: The mkmmconv tool detected a non-text character in <mmname>.i. The mkmmconv tool accepts the following characters in index files:

Visible character	ASCII 33 (!) through 126 (~)
Space	ASCII 32
Tab	ASCII 9
Newline (Linefeed)	ASCII 10
Carriage Return	ASCII 13 (^M)
Formfeed	ASCII 12 (^L)

Resolution: Remove or replace the offending characters from <mmname>.i and then rerun mkmmconv.

```
Error: No index file <mmname>.i.
```

Problem: The specified index file, <mmname>.i, does not exist.

Resolution: If the index file name is misspelled on the mkmmconv command line, correct it. Make sure you run mkmmconv in the directory containing <mmname>.i.

```
Error: mkmmconv interrupted.
```

Problem: mkmmconv was interrupted by the user.

Resolution: Rerun mkmmconv.

```
Warning: Cannot open warning file <mmname>.warn.
```

Problem: mkmmconv detected one or more problems during index file conversion but was unable to open <mmname>.warn to save the associated warning messages via a write operation. Since the program could not open <mmname>.warn, this warning does not get recorded to that warning file.

Resolution: This warning message usually indicates that you do not have privileges to create files within the directory in which mkmmconv was invoked (for instance, you may be on a read-only file system). It could also indicate serious disk space or system problems. Resolve the problem and then rerun mkmmconv.

```
Warning: Line <num> of <mmname>.i: \
Nonstandard PIB file <filename>.
```

Problem: The mkmmconv tool detected a **pib:** entry having a nonstandard PIB file field in <mmname>.i. This warning appears only when the **picky=y** option is used.

Resolution: The PIB file name either is not supported by Cabletron (which may be all right, as there are PIB files developed outside of Cabletron) or else the PIB file name is misspelled (which is a problem, as the **pib:** entry will not be properly processed). Carefully check the spelling of the PIB file name, correcting it if applicable. If the **pib:** entry is correct, you can suppress this warning message by running mkmmconv without the **picky=y** option.

```
Warning: Unknown option "<option>".
```

Problem: mkmmconv has been invoked with the illegal command line option <option>. This warning message is not recorded to the <mmname>.warn error file.

Resolution: Correct the offending command line option.

mkmm and *mkmmdeps* Errors

The following is a list of error messages, warning messages, and information messages that may be generated by the ***mkmm*** or ***mkmmdeps*** tool. They are listed together because many are shared by both tools. Any such messages will be displayed to the screen and also will be saved in files generated as <mmname>.err or <mmname>.warn, respectively, for subsequent viewing. The

text following each individual output explains what causes the given message — and, when applicable, outlines what you should do in response to that particular message.

```
Created MM archive <mmname>.tar
```

Problem: This message does not indicate an error. The `mkmm` tool has successfully created the extension module archive, identified as `<mmname>.tar`.

Resolution: `<mmname>.tar` may now be added to a manufacturing area using the `mkarea` tool, or installed directly into an existing SPECTRUM area using the `Install.quick` tool.

```
Error: Archival of MM <mmname> failed.
```

Problem: `mkmm`'s archiver failed while trying to create the extension module archive `<mmname>.tar`.

Resolution: `mkmm` probably ran out of disk space. Increase disk space and then rerun `mkmm`.

```
Error: Archival of product <prod> for MM <mmname> failed.
```

Problem: `mkmm`'s archiver failed while trying to create the subarchive for the product field `<prod>` of the `<mmname>` extension module.

Resolution: `mkmm` probably ran out of disk space. Increase disk space and then rerun `mkmm`.

```
Error: Cannot open error file <mmname>.err.
```

Problem: `mkmm` detected problems during the index file conversion, but was unable to open `<mmname>.err` to save the associated error messages via a write operation. Note that this error message, itself, cannot be written to the unopened `<mmname>.err` error file, so there will not be any record.

Resolution: This error message usually indicates that you do not have privileges to create files within the directory in which `mkmm` was invoked (for instance, you may be on a read-only file system). It could also indicate serious disk space or system problems. Resolve the problem and then rerun `mkmm`.

```
Error: File <mmname>.i: \
Absolute target directory in "<label>" entry.
```

Problem: The `<tdir>` field of a **`file:`** or **`head:`** entry is an absolute directory, which means that `Install` will extract the shippable file into an absolute directory rather than to a directory that is relative to the installation root directory. This condition must not be allowed, because this could enable `Install` (which runs as root) to extract files directly into system directories such as `/usr/lib` or `/usr/include`, thereby changing the system behavior of the host machine of the installation without the installer's knowledge.

Resolution: Change the `<tdir>` field of the **`file:`** or **`head:`** entry to a relative directory, so that the file is extracted from the medium within the installation root directory. If you must install a file within a system directory (which is not recommended), use a custom install script to move the file from its extracted location into the system directory. The custom script should confirm or warn that files are being installed in system directories, so that the installer can assent to, or at least know about, these installed system files.

```
Error: File <mmname>.i: \
Cannot create temporary directory "<dir>".
```

Problem: `mkmm` was unable to create the temporary directory `<dir>`, which it needs to be able to do in order to create the temporary files required to generate the extension module archive.

Resolution: This error message usually indicates that you do not have privileges to create directories within the directory in which `mkmm` was invoked (for instance, you may be on a read-only file system). It could also indicate serious disk space or system problems. Resolve the problem and then rerun `mkmm`.

```
Error: File <mmname>.i: Cannot find or execute <util>.
```

Problem: `mkmm` was unable to find or else could not execute the required support utility, `<util>`.

Resolution: This error probably indicates that the `mkmm` executable is not in its original location, as shipped within the SEI Toolkit, or that a symbolic link to the `mkmm` executable is being used. If this is the case, return the `mkmm` executable to its original directory within the SEI Toolkit and invoke it from that location.

It is also possible that files within the SEI Toolkit have been inappropriately modified or removed. In this case, reinstall the SEI Toolkit and then rerun `mkmm`.

```
Error: File <mmname>.i: \
Current platform (<os> <rev>) unsupported.
```

Problem: `mkmm` is not supported on the current host platform.

Resolution: Invoking `mkmm` on a given platform creates an extension module for that platform. Therefore, invoking `mkmm` on an unsupported platform would create an extension module archive for an unsupported platform, which would be uninstallable. Invoke `mkmm` only on the platform for which you intend to create the extension module.

```
Error: File <mmname>.i: Index file creation failed.
```

Problem: `mkmm` processes `<mmname>.i` to remove comments and other possibly sensitive information before shipping it in the extension module archive file. This error message signifies that `mkmm` was unable to create a temporary file for the modified index file.

Resolution: This error usually indicates disk space or system problems. Resolve the problem and then rerun `mkmm`.

```
Error: File <mmname>.i: Missing "<label>" entry.
```

Problem: `mkmm` has detected that a required entry with label `<label>` is missing from `<mmname>.i`. `<label>` will be one of **`mm:`**, **`rev:`**, **`irev:`**, **`level:`**, or **`vend:`**.

Resolution: Add the required entry to `<mmname>.i` and then rerun `mkmm`. The syntax and meaning of these entries is described in *Index File Entry Definitions*, starting on page 3-6.

```
Error: Line <num> of <mmname>.i: \
"<label>" entry has fewer than <num> fields.
```

Problem: The index file entry with label `<label>` requires at least `<num>` fields.

Resolution: Add the missing fields to the entry and then rerun `mkmm`. The proper syntax of all index file entries is detailed in *Index File Entry Definitions*, starting on page 3-6.

```
Error: Line <num> of <mmname>.i: \
"<label>" entry has more than <num> fields.
```

Problem: The index file entry with label `<label>` should have at most `<num>` fields.

Resolution: Remove the extra fields from the entry and then rerun `mkmm`. The proper syntax of all index file entries is detailed in *Index File Entry Definitions*, starting on page 3-6.

```
Error: Line <num> of <mmname>.i: \
"<label>" entry illegal in file <file>.
```

Problem: An index file entry other than **`deflt:`** appears in the `mkmm.sys` or `mkmm.loc` file used by `mkmm`. Only **`deflt:`** entries are allowed in these files.

Resolution: Remove the offending entry from `mkmm.sys` or `mkmm.loc` and then rerun `mkmm`. For more information on `mkmm.sys` and `mkmm.loc`, see the section entitled *mkmm.sys and mkmm.loc*, starting on page 4-11.

```
Error: Line <num> of <mmname>.i: \
"plat=" and "plat!=" flags on same line.
```

Problem: A single index file entry (such as **`file:`**, **`head:`**, or **`deflt:`**) has both **`plat=`** and **`plat!=`** modifiers; this condition is not allowed, because these modifiers are contradictory.

Resolution: Modify the index file entry to use either **`plat=`** or **`plat!=`** flags, but not both, and then rerun `mkmm`. For more information on these modifiers, look up the affected index file entry in the section entitled *Index File Entry Definitions*, starting on page 3-6.

```
Error: Line <num> of <mmname>.i: <sdir> not a directory.
```

Problem: A **`file:`** or **`head:`** entry contains an `<sdir>` field that does not specify an existing directory.

Resolution: Specify an existing directory in the <sdir> field of the **file:** or **head:** entry and then rerun mkmm. For more information on these entries, see the *Index File Entry Definitions* entries for *file:*, on page 3-18, or for *fdir:*, on page 3-21, as applicable.

```
Error: Line <num> of <mmname>.i: \
Cannot reset required default.
```

Problem: A **deflt:** entry is attempting to override a default target directory that cannot be overridden.

Resolution: Remove the offending **deflt:** entry and then rerun mkmm.

Certain types of files must be extracted from the medium into specific target directories; otherwise, Install may fail to properly install the files. For such file types, the default target directory (<tdir> field) is set in mkmm.sys, with the result that it cannot be overridden by **deflt:** entries or explicit <tdir> fields in **file:** or **head:** entries.

```
Error: Line <num> of <mmname>.i: \
Environment variable $var undefined.
```

Problem: Your index file contains a reference to the environment variable **$var**, but that variable was not exported prior to running mkmm.

Resolution: Export $var and then rerun mkmm.

```
Error: Line <num> of <mmname>.i: \
Illegal entry label "<label>".
```

Problem: An index file entry has the illegal label <label>.

Resolution: If <label> is misspelled, correct the problem and then rerun mkmm.

This problem may also be due to the presence of “old-style” index file entries in your index file. Many index file entries became obsolete as of SPECTRUM 4.0, and mkmm will no longer recognize the labels for those entries as legal. If this is the case, upgrade your index file to SPECTRUM 4.x/5.0 format, as described in the section entitled *Index File Conversion*, starting on page 3-37.

```
Error: Line <num> of <mmname>.i: \
Invalid default name "<dname>".
```

Problem: The index file contains a **`deflt:`** entry having an invalid <dname> field. The <dname> field must be one of `tdir` or `sdir`.

Resolution: Correct the <dname> field of the **`deflt:`** entry and then rerun `mkmm`.

```
Error: Line <num> of <mmname>.i: Invalid level "<level>".
```

Problem: The index file contains a **`level:`** entry containing an invalid <level> field. The <level> field must be one of **`0`**, **`1`**, or **`2`**.

Resolution: Correct the <level> field of the **`level:`** entry and then rerun `mkmm`.

```
Error: Line <num> of <mmname>.i: Invalid product "<prod>".
```

Problem: The index file contains a **`file:`**, **`head:`** or **`deflt:`** entry containing an invalid <prod> field. The <prod> field must be one of `Install`, SS, or SG.

Resolution: Correct the <prod> field of the **`file:`**, **`head:`** or **`deflt:`** entry and then rerun `mkmm`.

```
Error: Line <num> of <mmname>.i: \
No default source file name for "<entry>".
```

Problem: The <sname> field of a **`file:`** or **`head:`** entry is **`?`**, indicating a default value should be supplied. There is no default value for the <sname> field. For more information on these entries, see the *Index File Entry Definitions* entries *file:*, on page 3-18, or *fdir:*, on page 3-21, as applicable.

Resolution: Supply an explicit value in the <sname> field and then rerun `mkmm`.

```
Error: Line <num> of <mmname>.i: \
Non-flag "<field>" in wrong position.
```

Problem: An index file entry contains a non-flag field where only flag fields should appear. (For example, only flags should appear after the seventh field of a **`file:`** entry.)

Resolution: Correct the offending field and then rerun `mkmm`.

```
Error: Line <num> of <mmname>.i: Non-text character.
```

Problem: The `mkmm` tool detected a non-text character in `<mmname>.i`. `mkmm` accepts the following characters in index files:

Visible character	ASCII 33 (!) through 126 (~)
Space	ASCII 32
Tab	ASCII 9
Newline (Linefeed)	ASCII 10
Carriage Return	ASCII 13 (^M)
Formfeed	ASCII 12 (^L)

Resolution: Remove or replace the offending characters from `<mmname>.i` and then rerun `mkmm`.

```
Error: Line <num> of <mmname>.i: \
Repeated "<label>" entry.
```

Problem: An index file entry contains the label `<label>`, which should be unique but which appears two or more times in the index file. `<label>` will be one of **`mm:`**, **`rev:`**, **`irev:`**, **`level:`**, or **`vend:`**.

Resolution: Remove repeated entries and then rerun `mkmm`.

```
Error: Line <num> of <mmname>.i: \
Repeated "file: Install mmdesc" entry.
```

Problem: The index file contains two or more **`"file: Install mmdesc"`** entries, which will result in the shipment of two or more MM description files in the extension module. There can only be one MM description file shipped with each extension module.

Resolution: Remove repeated entries and then rerun `mkmm`.

```
Error: Line <num> of <mmname>.i: Target path name too long.
```

Problem: Because of limitations in the distribution software, files cannot be shipped or installed into directories having a path name exceeding a total of 100 characters. That is, for any **`file:`** or **`head:`** entry, `<tdir>/<tname>` cannot contain more than 100 characters.

Resolution: If possible, install files with shorter path names and then rerun `mkmm`. If you must install a file having a longer path name, ship the file with a shorter legal path name and then use a custom install script to move the file to the desired location.

```
Error: Line <num> of <mmname>.i: \
Unable to archive file "<file>".
```

Problem: The source file specified in a **`file:`** or **`head:`** entry (`<sdir>/<sname>`) is missing or unreadable. For more information on these entries, see the *Index File Entry Definitions* entries *file:*, on page 3-18, or *fdir:*, on page 3-21, as applicable.

Resolution: If the **`file:`** or **`head:`** entry is in error, correct it and then rerun `mkmm`. Otherwise, take measures to ensure that the specified file is accessible by `mkmm`.

```
Error: No index file <mmname>.i.
```

Problem: The specified index file, `<mmname>.i`, does not exist.

Resolution: If the index file name is misspelled on the `mkmm` command line, correct it and then rerun `mkmm`. Make sure you invoke `mkmm` from the directory containing `<mmname>.i`.

```
Error: mkmm interrupted.
```

Problem: `mkmm` was interrupted by the user.

Resolution: Rerun `mkmm`.

```
Error: mkmmdeps interrupted.
```

Problem: `mkmmdeps` was interrupted by the user.

Resolution: Rerun mkmmdeps.

```
Warning: Cannot open warning file <mmname>.warn.
```

Problem: mkmm detected warnings during the index file conversion, but was unable to open <mmname>.warn to save the warning messages via a write operation. Note that this warning also will not be recorded in the unopened <mmname>.warn. warning file.

Resolution: This warning message usually indicates that you do not have privileges to create files within the directory in which mkmm was invoked (for instance, you may be on a read-only file system). It could also indicate serious disk space or system problems. Resolve the problem and then rerun mkmm.

```
Warning: File <mmname>.i: \
Current platform "<plat>" undeclared.
```

Problem: <mmname>.i contains a **plat:** entry that does not list the platform on which mkmm is being run.

Resolution: Add <plat> to the **plat:** entry and then rerun mkmm.

```
Warning: File <mmname>.i: \
Missing "head: Install mmdesc" entry.
```

Problem: mkmm requires that every extension module ship an MM description file, which is shipped via a required **"head: Install mmdesc"** entry, but no such entry has been found.

Resolution: Create an MM description file and ship it in your extension module. The process for doing this is detailed in *MM Description Files*, starting on page 5-11.

```
Warning: Line <num> of <mmname>.i: \
Default target directory required.
```

Problem: An explicit <tdir> field in a **file:** or **head:** entry is attempting to override a default target directory that cannot be overridden.

Resolution: Change the offending <tdir> field to **?** (i.e., use the standard default target directory) and then rerun mkmm.

Certain types of files must be extracted from the medium into specific target directories; otherwise, `Install` may fail to properly install the files. For such file types, the default target directory (`<tdir>` field) is set in `mkmm.sys`, with the result that it cannot be overridden by **`deflt:`** entries or explicit `<tdir>` fields in **`file:`** or **`head:`** entries.

```
Warning: Line <num> of <mmname>.i: \
"file: SS db" entry requires "dep:" entry.
```

Problem: In the index file, there is a "**`file: SS db`**" entry for shipping a SpectroSERVER database import file. At installation time, this database import file must imported into the SpectroSERVER database in the correct order relative to database import files from other extension modules. To ensure this correct sequence, you must add one or more **`dep:`** entries to your index file to establish the proper ordering dependencies between your extension module and those other modules whose database import files must be imported first. For further information, refer to the **`dep:`** entry discussion on page 3-12.

Resolution: Add any appropriate **`dep:`** entries to your index file and then rerun `mkmm`.

```
Warning: Line <num> of <mmname>.i: "file:<prod><type>" \
entry inappropriate in level <level> index file.
```

Problem: As of SPECTRUM 4.0, certain types of files can not be shipped in extension modules of certain levels. The main offenders will be files for which `<type>` is `xtn` (which can no longer be shipped in level 0 or level 1 extension modules).

Resolution: Modify the offending **`file:`** or **`head:`** entry so that the `<type>` field is valid in an index file of level `<level>` and then rerun `mkmm`. Valid levels for each file type may be found in Table 5-4, *Special Processing of Shippable Files by Type*, on page 5-8. For files of type `xtn` in level 0 or level 1 index files, change the file type from `xtn` to `file`.

```
Warning: Line <num> of <mmname>.i: \
Possible flag "<field>" in wrong position.
```

Problem: `mkmm` has detected what it thinks is a flag `<field>` in a non-flag field position in an index file entry. `mkmm` identifies flags by the presence of an equals sign (**`=`**) in the field. This warning message appears only when the **`picky=y`** option is used.

Resolution: If the field is incorrect, correct it and then rerun `mkmm`. Otherwise, suppress this warning by running `mkmm` without the **`picky=y`** option.

```
Warning: Line <num> of <mmname>.i: \
Undeclared platform "<plat>".
```

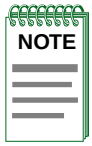
Problem: An index file entry uses a **`plat=<plat>`** or **`plat!=<plat>`** modifier, where **`<plat>`** has not been declared in a previous **`plat:`** entry.

Resolution: Include **`<plat>`** among the platforms listed in the **`plat:`** entry and then rerun `mkmm`. If necessary, create a new **`plat:`** entry. The **`plat:`** entry must precede any usages of **`plat=<plat>`** or **`plat!=<plat>`** modifiers in the index file.

```
Warning: Line <num> of <mmname>.i: \
Undefined product/filetype pair.
```

Problem: A **`file:`**, **`head:`** or **`deflt:`** entry has `<prod>` and `<type>` fields for which defaults have not been defined in `mkmm.sys` or `mkmm.loc`.

Resolution: The `<prod>` or `<type>` field of the **`file:`**, **`head:`** or **`deflt:`** entry is incorrect. Fix the incorrect field and then rerun `mkmm`.



You should not add `deflt:` *entries to* `mkmm.loc` *to define previously nonexistent file types.* `Install` *will not be able to properly install files of a file type defined in this way. If you wish to ship files that undergo special installation processing, use either* `<prod>=SS` *or* `<prod>=SG` *and then* `<type>=file` *and then use a custom install script to perform the special installation processing.*

```
Warning: Line <num> of <mmname>.i: \
Unsupported platform "<plat>".
```

Problem: A **`plat:`** entry, or a **`plat=`** or **`plat!=`** modifier, specifies a platform for which SPECTRUM distribution and installation are not supported.

Resolution: Edit your **`plat:`** entries, and **`plat=`** or **`plat!=`** modifiers so as to use only supported platform names and then rerun `mkmm`. Supported platform names are listed in Table 3-1, *Standard Platforms for SPECTRUM Extension Modules*, on page 3-6.

```
Warning: Unknown option "<option>".
```

Problem: `mkmm` has been invoked with the illegal command line option `<option>`. Note that this warning message does not get written to the unopened `<mmname>.warn` error file.

Resolution: Correct the offending command line option and then rerun `mkmm`.

Glossary

This glossary defines terms related to the manufacturing/distribution/integration processes for SPECTRUM extension modules.

This glossary file consists of three separate sections, each independently arranged in alphabetical order. The first section covers words pertaining to use of the SEI Toolkit. The second section is a general glossary of words used in common SPECTRUM terminology. The final section is a list of common SPECTRUM acronyms.

Glossary of Terms for SEI Toolkit

CD-ROM – Compact disc read-only memory. Major releases of SPECTRUM are now typically distributed on CD-ROM. Support for CD-ROM distribution of SPECTRUM software by VARs and customers is not currently available.

core component – A SPECTRUM extension module that supports basic SPECTRUM functionality. Core components are required by SPECTRUM and always integrated into SPECTRUM. Core components are developed exclusively at Cabletron.

custom component – A type of VAR-defined component that is not part of the SPECTRUM product (for example, external applications like protocol agents or gateways or a new type of SG-Support file). *See also standard component*.

custom install – The process of installing custom components. The SPECTRUM integration tool, **`Install`**, invokes any VAR-supplied custom install script or tools at a certain point in the integration process. *See also standard install*.

custom install script – Shell script files that perform various SEI Toolkit activities in the process of installing custom components. (*See Custom Install Scripts*, on page 5-13, for more information.) The SPECTRUM integration tool, **`Install`**, invokes any VAR-supplied custom install script at a certain point in the integration process.

database – In SPECTRUM, a set of several binary files that are used by SPECTRUM's database management system.

`dbtool` – A SPECTRUM utility that creates SPECTRUM database export (***.e***) files and imports (merges) the contents of externally prepared export files into the SPECTRUM database. The **`dbtool`** utility, which essentially is a command-line version of the Model Type Editor program, also provides reports concerning the contents of *.e files.

development area – The file systems and directory structure in which SPECTRUM extensions are developed.

disk tape – *See virtual tape*.

distribution – The process of packaging SPECTRUM software on a medium for later installation at a customer site. Also, any such medium containing SPECTRUM software.

distribution log file – A log file created by **`mkdist`**. The distribution log file contains a record of the actions performed by **`mkdist`** when it was last run.

distribution medium – *See medium*.

error log file – A log file created by **`mkmm`** if errors occur when **`mkmm`** is run. The error log file has an .err extension.

export – The process of extracting model type information from a source SPECTRUM database into an extract file (**`.e`** extension) using either **`dbtool`** or the Model Type Editor.

export file – An extract file created by **`dbtool`**, using the "export" option, or by the Model Type Editor. *See also extract file*.

extension – A modification to some aspect of SPECTRUM that provides additional capabilities or supports additional or specialized devices — whether customized for a particular customer site or developed for resale to the entire SPECTRUM customer base.

extension module – A software module that may be packaged to work with SPECTRUM. There are three types of extension modules: *core components*, *management modules*, and *external applications*. The term "extension module" more properly describes the latter two. An extension module can be device-specific (such as a Cabletron Bridge) or function-specific (such as a gateway). The packaging standards for SPECTRUM-compatible extension modules are implemented by the SEI Toolkit.

external application – A SPECTRUM extension module that is not integrated into SPECTRUM, but which contains tools that can be used within the SPECTRUM environment. External applications are often developed by customers or VARs and distributed using the SEI Toolkit.

extract file – A binary file, characterized by an extension of ".e", that is produced by either **`dbtool`** or the Model Type Editor (*see also*

export). It can then be used either by **dbtool** or by the Model Type Editor to add information to an existing SPECTRUM database (*see also* import).

import – The process of integrating model type information into a target SPECTRUM database from an extract file (.e extension) using either **dbtool** or the Model Type Editor.

import file – An extract file used with the **dbtool** "import" option or by the Model Type Editor. *See also extract file*.

index file – A text file used by the SPECTRUM packaging and integration process. Index files are written in a format specified by Cabletron. These files are interpreted by several tools in the packaging and integration process. Typically, each extension module or separately packaged product has its own index file. In this file, the exact contents of an extension module are spelled out. (Refer to Chapter 3, *Index Files*, for more information.)

***Install* –** The utility used for installing SPECTRUM software from a medium.

installation – The process of extracting SPECTRUM software from a medium and integrating the extracted software with existing SPECTRUM software at a customer site.

integration – The process of constructing a SPECTRUM Extension that can be installed at a SPECTRUM customer site using the standard SPECTRUM Installation script, **Install**.

log file – Files that contain a record of an activity performed by the SEI Toolkit. Refers to either of two types of logs created by the SEI Toolkit: error log files and distribution log files.

magnetic tape (physical tape) – As of SPECTRUM 5.0, the SEI Toolkit no longer supports magnetic tape. Developers wishing to distribute extensions should refer to *Virtual Tape (Vtape)*, starting on page 4-29, and/or to *Scenario 1: MM Vtape Distribution*, starting on page 6-1.

management module – A C++ coded software simulation of a physical device, using window-based icons to represent the device in a user interface and providing visual status through the use of various color definitions, together with access to in-depth information concerning the device's configuration and operating activity.

manufacturing area – A directory structure, defined by Cabletron, that contains all the necessary tools and files to make an extension module on the distribution media. All extension modules developed by a VAR coexist in this area when suitable for release. The tools provided allow creating custom distributions of any set of extension modules. *See also parts file*.

medium – Mechanism for storing SPECTRUM software for transport and installation at a customer site. The SEI Toolkit supports distribution of SPECTRUM software to two types of media: CD-ROM and virtual tape. *Also called distribution medium*.

mkarea – A tool that disperses components of the tar archive (created by **mkmm**) to respective directories under the manufacturing area.

mkdist – A tool that moves the SPECTRUM extensions specified in the parts file onto the desired distribution media.

mkmanf – A tool that creates the manufacturing area directory structure.

mkmm – A tool that makes an MM archive file from an index file.

MM – *See management module*.

MM archive file – A tar-formatted file consisting of a target index file and all product and log files for a particular management module (MM). An MM archive file is created during **mkmm** execution.

MM description file – A MM archive file includes a detailed description of the extension module, with this information being displayed at the time of installation to help the installer select the appropriate extension module. An MM description file is created during **mkmm** execution and is required in order for that program to complete. (For more information, refer to *MM Description Files*, on page 5-11.)

MM.tar file – *See* MM archive file.

parts file – A text file that indicates to **mkdist** which extensions to include on the distribution media. It contains a listing (by MM handle) of each extension module to be included in the distribution. (For more information, refer to *Manufacturing: mkdist*, on page 4-23.)

pib entry – An index file entry that must be removed, converted, and shipped in a PIB (Perspective Information Block) contribution file, which provides needed information to SPECTRUM's PIB database. For more information, refer to *PIB Contribution Files*, starting on page 5-30. (For more information on the purpose of the files in the PIB database, refer to the ***SPECTRUM IIB Editor Guide***.)

physical tape (magnetic tape) – As of SPECTRUM 5.0, the SEI Toolkit no longer supports physical tape. Developers wishing to distribute extensions should refer to *Virtual Tape (Vtape)*, starting on page 4-29, and/or to *Scenario 1: MM Vtape Distribution*, starting on page 6-1.

platform – Specific computer hardware (often proprietary) or combination of hardware and operating software having specific requirements and/or limitations for which application software must

be specifically designed — or which must be accommodated by the software in order for that software to be fully executed.

script – Shell script files that perform various SEI Toolkit activities. Most of the tools, such as **`Install`**, are shell scripts. The SEI Toolkit uses Bourne shell scripts.

SEI Toolkit – SPECTRUM Extensions Integration Developer's Toolkit. A SPECTRUM external application, consisting of a set of software tools, used for development, distribution, and installation of customer and VAR-developed extension modules for installation at customer sites.

SG-Support Files – Collectively refers to the ASCII text files used by SPECTRUM to determine how information is presented through SpectroGRAPH. These files define the various information blocks (such as Gibs, Pibs, and Iibs). These files are installed in the SG-Support directory.

SS Files – The files in the SS directory that make up the SpectroSERVER.

SG Files – The files in the SG directory that make up SpectroGRAPH.

stamp – A tool that prepares CsIib and CsGib files for distribution to a customer. This process adds a version "stamp" to each of the files related to an extension module under release. (For more information, refer to *Development: mkmm*, on page 4-7.)

standard component – Files such as SG-Support files (for example, **`CsPib`**, **`CsRib`**, and **`CsIib`**) and object files for inference handlers and model intelligence that are installed as part of the standard install when **`Install`** is run.

standard install – The installation of standard components by **`Install`**.

tape log file – A file created by **`mkdist`**, listing the contents of the distribution tape created by **`mkdist`**.

toolkit – Generally, a collection of software tools used for a common purpose, usually to construct new software. For SPECTRUM, two levels of toolkit products are available. Level I Tools provide the means to extend SPECTRUM without C++ coding, while Level II Tools facilitate the extension of SPECTRUM through developing new C++ code-based programs.

virtual tape – A disk-resident directory structure containing files organized as they would be on a physical tape; the name is an historical accident that has stuck. SPECTRUM extension modules can be distributed to and installed from a virtual tape. *Also called disk tape or* vtape. Developers wishing to distribute extensions should refer

to *Virtual Tape (Vtape)*, starting on page 4-29, and/or to *Scenario 1: MM Vtape Distribution*, starting on page 6-1.

vtape – *See* virtual tape.

General Glossary of Terms for SPECTRUM

The SPECTRUM documentation set contains a number of terms and acronyms specific to the SPECTRUM software product. This section provides descriptions for these terms and acronyms.

alarm, Alarm Register, Alarm Manager – Alarms occur when the VNM (Virtual Network Machine) detects a problem with a network model. The alarm is placed in the Alarm Register for the model, with the current status and probable cause of the alarm. The alarm is written to the event log (*which see*). The alarm can be examined by use of the Alarm Manager application until the alarm is cleared by the user (if possible) or changed during the polling process.

annotation – Text or simple line art that is placed in a view, generally to enhance or explain the information in the view. For SPECTRUM, annotations can be lines, circles, ellipses, rectangles, or text. These can be in various sizes, fonts, and colors.

association – The application of a relation rule in the SPECTRUM database. See *relation* and *rules*.

base model type / derived model type – Complex model types are derived from simpler model types. A complex model type may derive characteristics from several “parent” model types that provide their characteristics. The new model type is the “child,” having the combined characteristics of its “parents.” A base model type is a “parent” for derived model types. Each derived type is considered a “child.”

click / double-click – Common terms for mouse button operations. To “click” the mouse is to press and release the left mouse button once, while the mouse pointer is over the item that you wish to select.

> “Double-clicking” is a shortcut operation to give you access to icon features by pressing and releasing the left mouse button twice in rapid succession while the mouse pointer is over the item you wish to select. The double-click interval is the time window for making the two mouse clicks, as defined in the **`Spectrum/app-defaults/spectrum`** file or in an over-riding **`~/.Xdefaults`** file.

The functionality of the middle and right mouse buttons was changed effective with Release 4.0 in order to complement Windows NT conventions. For details, please consult *Appendix C* of the ***SPECTRUM Operator's Reference***.

connection pipe – In DevTop views, Cablewalk views and Topology views, logical connections are represented by images that are designed to look like pipes. These connections do not necessarily represent cables, but simply represent the fact that the icons are connected in some manner, with the pipes being color-coded in Release 5.0 to indicate operational status.

`.csi` file format – A "Cabletron Systems image" file format. Images must be in the .csi file format to be displayed in SPECTRUM. The .csi file format provides transparency and shadow features for SPECTRUM images.

datagram – A self-contained packet, independent of other packets in a data stream. A datagram carries its own routing information, and its reliable delivery does not depend on earlier exchanges between the source and destination devices. *See also frame* and *packet*.

derived model type – *see base model type / derived model type*.

device communication manager (DCM) – The DCM is a multi-protocol communication engine that handles communication with all network devices, regardless of their protocol. The DCM translates the SpectroSERVER polling into the protocol understood by the individual devices.

dialog box – A separate window that contains controls or selections that are needed only occasionally. It is also used to prompt the user for required information. There are two types of dialog boxes, modal and non-modal. *Non-modal* dialog boxes can remain open while performing tasks outside the box. *Modal* dialog boxes must be closed before performing any additional tasks.

drag – Movement of an item being held by the mouse to a new location. To do this, move the pointer onto the item, press the left mouse button, and hold that button depressed while you move the pointer (and the item) to the new location, and then release the mouse button.

dynamic – For SPECTRUM, a feature is dynamic if it changes to match the current situation. Menus and windows are dynamic in that their contents change to match the current mode, device, status, or other criteria.

event, event log – Significant messages from the VNM (Virtual Network Machine) are events. These are recorded in the Event Log Database for future viewing through the Event Log Window of SpectroGRAPH. Examples of events include device created, device destroyed, loss of contact with device, and VNM started.

Find Information Block (FIB) – A file that stores parameters for controlling the Find View feature.

frame – A frame is a data packet plus the rest time between packets. A minimum rest time of 9.6 µsec (microseconds) is required between frames. *See also datagram* and *packet*.

generic information block (GIB) – The GIB file stores parameters for controlling the "generic" screen views in SPECTRUM. A generic view displays a model's configuration, together with diagnostic and performance information. Addition and subtraction of data fields in a GIB file adds or subtracts the fields from every member of that model "class." Modifying the content of existing data fields of a GIB file generally affects only the specific model being operated on.

giant packet – *See jabber* and *packet*.

icon – A graphical design that represents a software model. Icons can have features that give the user access to information that is available on the model, such as statistics and configuration.

Icon Information Block (IIB) – The IIB file stores the Icon Manager's classes, raster file names, and the children icons with screen positions to be used for display.

icon manager – The software that controls icons on the user's on-screen display.

inferred model – A model that contains only internal attributes, whose values are inferred from other models' attributes.

inference handler – A portion of the C++ code providing intelligence to the SpectroSERVER. An inference handler performs a single, specific task.

instantiate – Creating a particular "instance" or specific occurrence of something.

jabber – A packet that exceeds the maximum allowable size. *See* Packet.

landscape – The models contained in your SPECTRUM database, together with the server that manages them.

landscape handle – A hexadecimal number uniquely identifying each model type, relation, model, attribute, and landscape.

MAC address – *See physical address*.

Mail Service – The Mail Service (Mail Gram or Mail System) provides a way for Icon Managers to communicate with the models that they represent.

model – A SpectroSERVER representation of a network device or network group that is being managed, such as a device, cable, port, etc. *See also model type*.

model type – A class of model. A template describing attributes, actions, and associations. *See also model* and *model type derivation* derivation.

model type derivation – The structure, or family tree, that describes how model types inherit their attributes. See also *model*.

multi-threaded application – An application that is capable of processing many operations simultaneously.

object-oriented design – A study of programming design involving encapsulated structures and methods. It allows you to create abstract objects that have a direct relationship to concrete objects.

octet / octet string – An octet is an eight bit word. An octet string is a series of octets grouped together.

off page reference icon – A topology icon for a device that cannot be viewed currently, but the device is directly connected to the current topology view in another level of detail.

packet – A group of bits transmitted together that includes a preamble, a source and destination address, a type/length field, data, and control elements, such as a Frame Check Sequence (FCS). *See also datagram* and *frame*.

pcx file format – A graphics "paint" program file format.

physical address – An association with a specific network interface (such as an interface card). Replacing the network interface would change the physical address for that node. A physical address is six octets (48 bits). Physical addresses are generally assigned at the point of manufacture for the network interface.

pixel – A "dot" on your video display monitor's screen. Pixels are arranged in patterns to create images on your screen. Refer to the documentation that came with your video display monitor for more information.

polling interval – In milliseconds, the time between polls of the network for a specific model by the VNM (Virtual Network Machine). *See also timing interval*.

polling ratio – The number of polls by the VNM (Virtual Network Machine) of a device on the network that will occur prior to logging the poll results in the database. For example, with a poll to log ratio of "10," the statistics gathered will be logged in the database once for every ten VNM polls of the device on the network.

popup menu – An independent menu that is related to something selected on the window. A popup menu duplicates functions available through the menu bar, but is more convenient for experienced users.

post / pick – To "post" a menu is to retain a copy of a menu on the screen. To "pick" from the menu is to select one of the menu options. Option selections are made from a posted menu by clicking on the option with the left mouse. A user posts a menu and picks an option from the menu.

pull-down menu – A menu panel that is accessible through a menu bar item.

raster – The background "artwork" for a SPECTRUM window. In Topology views and location views, the raster can provide information such as the layout of a building — or a map indicating a series of locations. Raster files may be in the **`.csi`** or **`.tif`** file format to be displayed in SPECTRUM.

redundancy – A networking technique to reduce network down time. Redundant communication paths are installed in the network, and a means of enabling the "spare" path is established.

relation – A relation is a classification describing how entities relate to each other. Each relation contains a list of "rules," which apply the relation to model types. Relations include, but are not restricted to: Encompasses, Passes_Through, Adjacent_to, Contains, Collects, Connects_to, Executes, and Monitors. *See also rules* and *association*.

rules – Applying a relation to specific model types is a rule. For example, in the Contains relation, "Building contains Room" is a rule saying that buildings can contain rooms. *See also relation* and *association*.

runt packet – Any packet that is smaller than the minimum packet size of 64 bytes long, not including the preamble.

screen – Refers to the complete display surface of the video monitor.

segment – A media link in a network connecting nodes together.

socket – An addressable model within a network node that is owned by a software process within the node. The software process is known as a socket client.

tagged image file format (TIFF) – Consists of a header and one or more Image File Directories (IFD). SPECTRUM supports compressed and uncompressed TIFF image files.

tear-off menu – A pull-down menu that can be "torn off" from its normal position on the menu bar and placed in a new location on the screen. This is convenient when a particular menu is accessed often and the user would like to keep the menu open for a period of time. When you are done, the tear off menu can be closed.

timing interval – In milliseconds, the time between GUI updates of the on-screen data. This value sets the SpectroGRAPH poll of the database. This is *not* the same as polling interval. *See also polling interval*.

user interface (UI) – Known as SpectroGRAPH for SPECTRUM. It provides all the interaction with the user and controls the screen, keyboard, and mouse, providing the views of the network model. The UI is also known as the graphical user interface (GUI).

view – One of many representations of the network model. Views can show physical locations (Location views), network layouts and connections (Topology Views), statistic and attribute values (the GIB views), or status (Alarm view and Event Log view).

virtual network machine (VNM) – The VNM is the intelligent software in SpectroSERVER.

wildcard – A character that replaces a range of characters, used primarily when filtering lists of files.

window – A window is a visible work space on the monitor screen, separately controllable from any other such spaces (which may be overlapping).

window manager – The window manager is a program to permit the user to alter and control the configuration of the windowing system on a display screen. For example, a window manager may provide the ability to resize and move a window.

Common SPECTRUM Acronyms

The following list identifies a number of acronyms that are unique to SPECTRUM or that have special meanings in SPECTRUM, as listed.

Acronym	Definition
.csi	Cabletron Systems Image (used as a file extension)
CDR	Compact Disk Recorder
DB	Database
DCM	Device Communications Manager for SpectroSERVER
ELM	Event Log Manager
FIB	Find Information Block
GIB	Generic Information Block
IH	Inference Handler
IIB	Icon Information Block
IMT	Inductive Modeling Technology
MI	Model Intelligence
MIM	Media Interface Module
MM	Management Module
MMD	Management Module Description
MTE	Model Type Editor
PIB	Perspective Information Block
RIB	Report Information Block
SEI	SPECTRUM Extensions Integration
SG	SpectroGRAPH
SS	SpectroSERVER
UI	The SpectroGRAPH User Interface
VFILE	Vendor file (created by non-Cabletron developer)
VNM	The SpectroSERVER Virtual Network Machine

Index

Symbols

##headersize directives 5-19
##include directive (discontinued) 5-23
##include directive (discontinued) 5-19
${CS_XAPPL_ICONBLINK} variable 5-27
${DEVAREA} environment variable 3-28
${DS} variable 5-27
${SG_ROOT} variable 5-27
${SG_SUP_ROOT} variable 5-28
${VAR} environment variable 5-27
${WINDOWING_SYS_LIB} variable 5-27
$AUTO variable 5-20
$AWKUTIL variable 5-21
$CPUTYPE variable 5-20, 5-21
$DEV_NULL variable 5-18, 5-22
$DEVAREA environment variable 6-2
$IROOT variable 5-22
$LD_LIBRARY_PATH variable 5-22
$LOG environment variable 5-25
$LOGDIR environment variable 5-25
$MANF environment variable 4-15, 4-16, 4-23, 6-11
$PATH variable 5-22
$RANLIB variable 5-22
$SG_ROOT variable 5-23
$SG_SUP_ROOT variable 5-23
$SG_TOOLS_ROOT variable 5-23
$SS_ROOT variable 5-23
$SS_TOOLS_ROOT variable 5-23
$WINDOWING_SYS_BIN variable 5-25
$WINDOWING_SYS_LIB variable 5-25
$WINDOWING_SYS_LIB_APP variable 5-25
$WINDOWING_SYS_TKLIB variable 5-25
(sun4c) 3-6
*.i file 3-27
*.i files 4-4
*.mmd file 3-27
*.p files 4-4
<defname> field 3-16
<file> argument 4-25
<file> field 3-15
<level> field 3-7
<mmname> argument 4-3, 4-5, 4-6, 4-21
<mmname> field 3-10, 3-12, 4-7, 4-9
<mmname>.err files 4-9, A-2, A-6
<mmname>.i file A-2
<mmname>.i.conv index file A-2
<mmname>.p contribution file A-2
<mmname>.tar extension module archive 4-9
<mmname>.tar file A-7
<mmname>.warn files 4-9, A-2, A-6
<mode> field 3-15, 3-22
<name> field 3-9
<option> option A-6
<optlist> option 4-33
<plat> field 3-22
<platlist> field 3-22
<prod> field A-7
<prod> field 3-16, 3-18, 3-21, 3-22, 5-20
<sdir> field 3-18, 3-19, 3-21, 3-22, 5-17, 5-20
<sname> field 3-18, 3-19, 3-22, 5-20
<snamepat> field 3-21
<tdir> field A-8
<tdir> field 3-18, 3-19, 3-21, 3-22, 3-29, 5-20
<tname> field 3-18, 3-19, 3-21, 3-22, 5-20
<type> field 3-16, 3-18, 3-21, 3-22, 5-20
<value> field 3-16
<vendor> field 3-10
<version> field 3-7, 3-9
"?" fields 3-17
"?" variable 3-33, 3-35
"check" option 2-4

Numerics

100-character limit in path name A-14

A

add option 4-21
app_def_set function 5-24
app-defaults file 5-26
archive files 2-4

assigning default values to installation parameters 5-20
attr: label 3-15
auto mode (Install) 5-18
awk program 5-21

B

backup_app_defs function 5-24
backward compatibility 1-2, 4-2
Bourne shell 3-28, 3-35, 5-14, 5-18, 5-19, 5-23, 6-10
  commands 3-24
  variables in custom install scripts 5-20
braces (used for expansion) 3-24
building SPECTRUM extension tar files 4-7

C

C++ 3-9, 6-5
Cabletron reserved index file entries 3-5
CDR (compact disk recorder) 4-31
CD-ROM
  format 4-29
  physical 4-29
  virtual tape (CD-ROM Vtape) 4-30
  virtual tape (CD-ROM vtape) 4-29
check option 4-10
check=min argument 4-23
check=print argument 4-24
check=y/n argument 4-10
checksum program, definition of 5-8
cleanup option (retired) 4-27
clobber=y/n argument 4-23, 4-24
command-line options
  for Install 4-33
  for mkdist 4-23
  for mkmm 4-8
comments in index files 3-2
comp: dependency (one per line) 3-11
comp: label 3-10, 3-27
comp=y/n argument 4-23, 4-24
compact disk recorder (CDR) 4-31
compress argument (retired) 4-20
compress gzip argument (retired) 4-20
compress= argument (retired) 4-20
conversion
  index files 3-37 to 3-42
  need for 1-2
  of unsupported labels A-5
convert=y/n argument 4-10
converted index files 4-4
converting empty file(s) A-4
converting nonexistent files A-5
core option (retired) 4-28
CsGib files 2-4
CsIib files 2-4
custom component, definition of Glossary-1
custom install scripts 1-3, 5-13 to 5-30
  extraction of 5-13
  labels for 6-5
  LD_LIBRARY_PATH variable 5-22
  location 5-17
  predefined environment variables 5-20 to 5-29
  purpose 5-17
  retired variables 5-24
  shipping 5-19
  testing on platforms 5-18
  using Bourne shell variables 5-20
custom installation, definition of Glossary-1
custom mode (Install) 5-18
custom script
  responsibilities 5-15

D

data item formats 3-6
database, definition of Glossary-1
dbtool 6-4, Glossary-2, Glossary-3
  file_name parameter 4-15
  parameters 4-15
deflt: label 3-16, 3-33, 3-34
deliver_app_defs function 5-24
dep: dependency (one per line) 3-13
dep: label 3-12, 3-27
dep=y/n argument 4-23, 4-24
depend=y/n argument 4-10
dependencies
  comp: (one per line) 3-11
  dep: (one per line) 3-13
devarea area 6-9
developer's tools integration 6-9 to 6-12
development area 2-3, 6-10
  definition of Glossary-2
development environment 2-3
development phase 2-3 to 2-4
  overview 2-2
  tools specifications 4-2 to 4-15

development tools specifications 4-2
device_name=<value> argument 4-33
disk space A-7
distribution
log file, definition of Glossary-2
media 4-29
creating 4-23
MM tape 6-1 to 6-9
shippable files 5-5 to 5-7
tape, extracting files from 4-32
doc (document) file 3-32
dollar sign ($) character 3-3

E

effect of non-text characters A-13
enforcing mode and file ownership 3-15
entries
extension module
description type 3-6 to 3-10
relation type 3-10 to 3-14
file: 5-13
level: 4-24
pib: 4-4
environment variables
forms–$VAR, ${VAR}, or $(VAR) 3-3
See variables
error and warning files
with mkmm 4-9
with mkmmconv 4-4
error log file A-1
error log file, definition of Glossary-2
error messages
mkmm 4-10
with Install.quick 4-18, 4-34
with mkdist 4-18, 4-27
with mkmm 4-10
with mkmmconv 4-4
with mkmmdeps 4-7
errors A-1
mkmm A-6
mkmmconv A-2
mkmmdeps A-6
errors vs. warnings A-1
examples
creating CD-ROM vtape 4-31
custom install script 5-17
extension module
description labels 3-6 to 3-10, 3-25 to 3-26
file distribution
label entries 3-26 to 3-37
source index file format 3-23
updating MMML file 4-19
vtape creation 4-29
writing custom install scripts 5-16
exit status
0 (zero) 4-4, 4-7, 4-10, 4-23
1 (one) 4-4, 4-7, 4-10, 4-23
for custome install scripts 5-19
Install.quick 4-34
mkarea 4-23
mkdist 4-27
mkmm 4-10
with >mkmmcconv 4-4
with mkmmdeps 4-7
export file
definition of Glossary-2
export process
definition of Glossary-2
distribution 5-5, 6-8
setting up 6-2
extension components, extracting 4-20
extension module xi
archives 4-22
description (examples) 3-25
description labels 3-6 to 3-26
examples 3-25 to 3-26
effect of installing in wrong order 3-14
level 4-24
MM levels 3-9
extension module archive file 4-9
extension tar file, building 4-7
extensions ix
definition of 1-2, Glossary-2
development of 2-1
placing on distribution media 4-23
Extensions Integration Toolkit xi
extract files, definition of Glossary-2
extract option 4-21
extracting distribution tape files 4-32
extracting extension components 4-20
extraction of custom install scripts 5-13
exts=<extlist> argument 4-5, 4-6, 4-7, 4-8

F

fdir: label 3-21
fdir: SG iib entry 3-36

field
<defname> 3-16
<file> 3-15, 4-25
<level> 3-7
<mmname> 3-10, 3-12, 4-3, 4-5, 4-6, 4-7, 4-9, 4-21
<mode> 3-15, 3-22
<name> 3-9
<option> A-6
<optlist> 4-33
<plat> 3-22
<platlist> 3-22
<prod> 3-16, 3-18, 3-21, 3-22, 5-20
<sdir> 3-18, 3-19, 3-21, 3-22, 5-17, 5-20
<sname> 3-18, 3-19, 3-22, 5-20
<snamepat> 3-21
<tdir> A-8
<tdir> 3-18, 3-19, 3-21, 3-22, 3-29, 5-20
<tname> 3-18, 3-19, 3-21, 3-22, 5-20
<type> 3-16, 3-18, 3-21, 3-22, 5-20
<value> 3-16
<vendor> 3-10
<version> 3-7, 3-9
add 4-21
check 4-10
check=min 4-23
check=print 4-24
check=y/n 4-10
clobber=y/n 4-23, 4-24
comp=y/n 4-23, 4-24
compress (retired) 4-20
compress gzip (retired) 4-20
compress= (retired) 4-20
convert=y/n 4-10
dep=y/n 4-23, 4-24
depend=y/n 4-10
device_name=<value> 4-33
extract 4-21
exts=<extlist> 4-5, 4-6, 4-7, 4-8
install_type=<value> 4-33
level=0/1/2 4-23
link=y/n (retired) 4-28
log=y/n (retired) 4-28
media_host=<value> 4-33
media_type=<value> 4-33
mm (retired) 4-28
mm=<mmname> 4-2, 4-3, 4-5, 4-6, 4-7, 4-21
multiple use 4-8
mmname 4-2
mode=<mode> 3-15, 3-18, 3-19, 3-21
nolog (retired) 4-28
parts 4-21, 6-8
picky=y/n 4-2, 4-3, 4-5, 4-7, 4-8
plat!=<plat> 3-17
plat=<plat> 3-16, 3-17, 3-18, 3-20, 3-21
plist=<file> 4-21, 4-23, 4-25
plist=partlist 6-8
prod=SS/SG 4-23
prompt (retired) 4-28
prompt=y/n (retired) 4-28
prot=yes 3-16, 3-18, 3-20, 3-21
remmm=y/n 4-23, 4-25
remote=y/n (retired) 4-28
remotetape (retired) 4-28
req=yes 3-16
rhost=<host> 4-28
sg_install=<value> 4-33
SpectroGRAPH (retired) 4-28
SpectroSERVER (retired) 4-28
ss_install=<value> 4-33
ss_link=<value> 4-33
tape=<dev> 4-29
tar=<dir> 4-22
tmp=<dir> 4-22
uid=root 3-15, 3-18, 3-20, 3-21
verbose (retired) 4-28
verify=y/n (retired) 4-28
vtape <dir> 4-28
vtape=<dir> 4-23, 4-26
x (retired) 4-28
xtn (retired) 4-28
xtn_install=<value> 4-33
fields
effect of having too many A-4
effect of not having enough A-4
file: entry 5-13, A-8
file: label 3-18
file: SS cus entry (example) 3-31
file: SS db entry (example) 3-29
file: SS doc entry (example) 3-32
file: SS ih entry (example) 3-30
file_name parameter
dbtool 4-15
files
*.p 4-4
distribution
examples 3-25
label entry examples 3-26
labels 3-15 to 3-23

error and warning 4-4
with mkmm 4-9
extension module archive 4-9
index 4-9
MM description 5-11 to 5-12
mode enforcement 3-15
must import in correct order 3-14
output 4-7
ownership enforcement 3-15
relocation requirements 5-15
requirements for moving 5-15
shippable 4-6, 4-9
statistics by product and type 5-6
files needing special installation processing A-17
foldover lines 3-2
format
CD-ROM vtape 4-29
functions
for shipping files 3-5

G

get_entries function 5-24
graphical applications 5-26
standards 5-29
grep 3-21, 5-18

H

head: entry A-8
head: label 3-22

I

identifying mkdist dependencies 3-10
ih (inference handler) file 3-30
import file, definition of Glossary-3
import process, definition of Glossary-3
importing files in correct order 3-14
inability to open file(s) via write operation A-2
include files 5-23
incompatibility with earlier versions 1-2
index file 2-4
changed format for 4.0 4-2
conversion 3-37 to 3-42
converted 4-4
definition of Glossary-3
definitions 3-6 to 3-23
entries 3-5, 3-6 to 3-23
"new-style" 3-5, 3-38
"old-style" 3-5, 3-38
Cabletron reserved 3-5
three types 3-5
label 3-5
legal characters 3-2
line, interpretation of 3-3
mkmmdeps 4-6
optional comments 3-2
platforms 3-5
syntax 3-23
syntax rules 3-1 to 3-4
upgrading preexisting files 4-2
usage rules 3-23
inference handlers (ih) nodes 6-5
info: label 3-23
informative messages A-1
input files
Install.quick 4-33
mkarea 4-22
mkdist 4-26
mkmanf 4-17
mkmm 4-9
mkmmconv 4-3
mkmmdeps 4-6
with mkmmdeps 4-6
insdk.cus 1-3
Install 1-3, 2-5, 3-7, 3-8 to 3-14, 4-29, 4-32, 5-14, 6-5, 6-8
auto mode 5-18
command-line options 4-33
custom mode 5-18
description 4-32
inability to handle dependencies across multiple installation tapes 3-14
interaction with dep: 3-12
interactive mode 5-18
noninteractive mode 5-18
usage 4-32
use of extension module level value 3-8
Install mmdesc entry 3-27
install scripts
writing 5-16 to 5-19
Install tool 2-3
Install.quick 2-3, 2-5, 4-32
description 4-32
error messages 4-18, 4-34
exit status 4-34

input files 4-33
location, invoking 4-33
output files 4-34
usage 4-32
install_type=<value> argument 4-33
installation of shippable files 5-8 to 5-10
installation parameters
assigning default values 5-20
installation services 5-14
installation support software 4-29
installed files
effect of installing in wrong order 3-14
moving 5-15
installing X resource files 5-15
integration
definition of Glossary-3
integration of developer's tools 6-9 to 6-12
integration phase 2-5
overview 2-3
tools specification 4-32 to 4-34
intel platform (Microsoft) 3-6
intelligence 6-5
interactive mode (Install) 5-18
interpretation of index file line 3-3
invalid label A-5
irev: label 3-7, 3-26
effect of omission 3-7
ISO 9660 format standard 4-31

K

Korn shell (ksh) 5-18

L

label
attr: 3-15
comp: 3-10, 3-27
deflt: 3-16, 3-33, 3-34
dep: 3-12, 3-27
effect of invalidity A-5
effect of misspelling A-5
effect of non-text characters A-5
fdir: 3-21
file: 3-18
formats 3-6
head: 3-22
info: 3-23
irev: 3-7, 3-26
level: 3-7, 3-26
mm: 3-9, 3-25
mtype: 3-23
plat: 3-22, 3-26
rev: 3-9, 3-26
vend: 3-10, 3-25
labels
Cabletron reserved type 3-23
extension module
description type 3-6 to 3-10
relation type 3-10 to 3-14
extension modules 3-6 to 3-14
file distribution type 3-15 to 3-23
for custom install scripts 6-5
LD_LIBRARY_PATH variable 5-22
length of entries 3-2
level: entry 4-24
level: label 3-7, 3-26
level=0/1/2 argument 4-23
line length 3-2
link=y/n argument (retired) 4-28
log files
definition of Glossary-3
log=y/n argument (retired) 4-28

M

magnetic tape
shipment of vtape 6-8
Makefile dependencies 4-5
management module 5-13
(MM file) ix
tar files (*See* MM.tar files)
MANF directory 6-11
manufacturing area 6-8, 6-11
definition of Glossary-3
files 4-22
setting up directory structure 4-15
manufacturing phase 2-4
overview 2-3
manufacturing phase tools
specifications 4-15 to 4-28
manufacturing tools specification 4-15
media
distribution 4-29
media_host=<value> argument 4-33
media_type=<value> argument 4-33
Microsoft platform (intel) 3-6
missing fields A-10
missing source file A-14

mkarea 2-3, 4-20, 4-26, 6-7, 6-12
  definition of Glossary-4
  description 4-20
  exit status 4-23
  input files 4-22
  location, invoking 4-20
  output files 4-22, 4-27
  system environment 4-20
mkdist 2-3, 4-23 to 4-28, 6-7, 6-12, Glossary-2
  command-line options 4-23
  definition of Glossary-4
  description 4-23
  error messages 4-18, 4-27
  exit status 4-27
  fails without **vtape=<dir>** option 4-26
  input files 4-26
  location, invoking 4-23
  output files 4-27
  usage 4-23
mkmanf 2-3, 4-15, 6-7, 6-11
  definition of Glossary-4
  description 4-15
  environment variables 4-16
  input files 4-17
  output files 4-17
  system environment 4-16
mkmm 2-2, 3-1, 3-4, 3-5, 3-7, 3-40, 4-3, 4-7 to 4-15, 6-6, 6-10, A-4
  command-line options 4-8
  definition of Glossary-4
  description 4-7
  environment variables 4-8
  error and warning files 4-9
  error messages 4-10
  errors A-6
  exit status 4-10
  input files 4-9
  location, invoking 4-8
  output files 4-9
  shippable files 4-9
  syntactical requirements 3-24
  system environment 4-8
  usage 4-7
mkmm.loc 4-6, 4-9
mkmm.sys 4-6, 4-9
mkmmconv 4-2, 4-3, A-2, A-4
  description 4-2
  error and warning files 4-4
  error messages 4-4
  errors A-2
  exit status 4-4
  input files 4-3
  location, involing 4-3
  output files 4-3
  system environment 4-3
  usage 4-2
mkmmconv 3-39
mkmmdeps 3-4, 4-3, 4-5 to 4-7
  description 4-5
  environment variables 4-5
  environment variables 3-6
  error messages 4-7
  errors A-6
  exit status 4-7
  index files 4-6
  input files 4-6
  location, invoking 4-5
  output files 4-7
  shippable files 4-6
  system environment 4-5
  usage 4-5
MM description files 5-11 to 5-12
  shipping 5-12
  writing 5-11
mm option (retired) 4-28
MM tape distribution 6-1 to 6-9
MM.tar files 2-4
mm: label 3-9, 3-25
mm=<mmname> argument 4-2, 4-3, 4-5, 4-6, 4-7, 4-21
  multiple use 4-8
MM_i directory 4-22, 4-27
MM_sg directory 4-22
MM_sg directory 4-27
MM_ss directory 4-22, 4-27
MMML (Master Management Module List) file 4-18
mmname argument 4-2
mode
  enforcing 3-15
mode=<mode> argument 3-15, 3-18, 3-19, 3-21
model intelligence (mi) nodes 6-5
model type
  handles 6-4
  list files 6-4
modifying contents of installed files 5-14
moving installed files 5-15
  requirements 5-15

mtype: label 3-23
multiple-name entries 3-39
mutually exclusive fields 3-17
myparts file 4-31

N

new-style index file entries 3-5
    (tabulation) 3-38
nolog option (retired) 4-28
noninteractive mode (Install) 5-18
nonstandard PIB file field A-6
non-text characters A-5, A-13
Notice i
nul: (standard output sink device in
        Windows NT) 5-19
NuTCracker Korn Shell 4-2

O

old-style index file entries
    (tabulation) 3-38
    replacement by new-style 3-5
option
    cleanup (retired) 4-27
    core (retired) 4-28
output files 4-7
    Install.quick 4-34
    mkarea 4-22, 4-27
    mkdist 4-27
    mkmanf 4-17
    mkmm 4-9
    mkmmconv 4-3
    mkmmdeps 4-7
ownership, enforcing 3-15

P

partlist file 6-8
parts file, definition of Glossary-4
parts list file 4-26
parts option 4-21, 6-8
path name limit (100 characters) A-14
Perl 3-21, 5-18
physical CD-ROMs 4-29
PIB contribution files 3-41, 4-4, 5-30
    automatic generation by mkmmconv 3-41
    manual generation 5-30
    updating 5-15
pib: entries 3-41, 4-4, 5-30
picky=y/n argument 4-2, 4-3, 4-5, 4-7, 4-8
plat!=<plat> plat!=<plat> argument 3-17
plat: label 3-22, 3-26
plat=<plat> argument 3-16, 3-17, 3-18,
        3-20, 3-21
platform
    dependendency 6-2
    distribution and installation 3-5
    index files 3-5
    intel 3-6
    standard names 3-5
    sun4c 3-6
    testing of custom install scripts 5-18
    Windows NT 4-31
plist=<file> argument 4-21, 4-23, 4-25
plist=partlist argument 6-8
portable installation support utilities 5-18
pound sign (#) character 3-3
predefined environment variables for custom
        install scripts 5-20 to 5-29
pre-mastering 4-31
privileges
    effect of not having A-3, A-6
processing of shippable files 5-8
prod=SS/SG argument 4-23
prompt option (retired) 4-28
prompt=y/n argument (retired) 4-28
prot=yes argument 3-16, 3-18, 3-20, 3-21
purchasable part 4-18
    names 4-19

Q

quick install 4-32

R

read-only file system A-8
recordable CD-ROM 4-31
records 5-13
release number (SPECTRUM) 1-1
relocation of files
    requirements 5-15
remmm=y/n argument 4-23, 4-25
remote=y/n argument (retired) 4-28
remotetape option (retired) 4-28
req=yes argument 3-16
required support utility, <util> A-8

Restricted Rights Notice ii
retired custom install script variables 5-24
retired features
of `mkdist` 4-27
of `mkmanf` 4-20
returns
exit status
0 (zero) 4-4, 4-7, 4-10, 4-23
1 (one) 4-4, 4-7, 4-10, 4-23
`rev:` label 3-9, 3-26
`rhost=<host>` argument 4-28

S

script
definition of Glossary-5
See also custom install scripts
search path 3-40
SEI process (Illust.) 2-2
SEI Toolkit
benefits 1-3
definition of Glossary-5
development phase 2-3 to 2-4
overview 2-2
goals 1-3
integration phase 2-5
overview 2-3
manufacturing phase overview 2-3
overview 1-2
purpose 1-3, 2-4
tools 4-2
tutorial 6-1 to 6-12
version compatibility 4-1
SG files
definition of Glossary-5
`SG iib` entry 3-28
`sg_install=<value>` argument 4-33
SG-Support 3-28
files 6-4
definition of Glossary-5
shippable files 4-6, 4-9, 5-1
development 5-3 to 5-5
distribution 5-5 to 5-7
file types 5-1 to 5-2
installation 5-8 to 5-10
levels 5-1 to 5-2
preferred development methods 5-3
products 5-1 to 5-2
relaltion to product 5-2
special processing 5-8
with `mkmm` 4-9
with `mkmmdeps` 4-6
shipping
custom install scripts 5-19
files that need special installation processing A-17
MM description files 5-12
Solaris
platform types 3-6
Sun Microsystems platform 3-6
source control area 3-42, A-2
source file
effect of missing A-14
extension handling 4-13
source index file format
examples 3-23
special characters
escaping 3-3
special processing of shippable files 5-8
special requirements 4-1
special shippable files 5-11 to 5-32
specifications
development tools 4-2
integration tools 4-32
manufacturing tools 4-15
`SpectroGRAPH` option (retired) 4-28
`SpectroSERVER` option (retired) 4-28
SPECTRUM
distribution
CD-ROM 4-29
CD-ROM vtape 4-31
extensions
definition of 1-2, Glossary-2
release number 1-1
version number 1-1
SS files
definition of Glossary-5
`ss_install=<value>` argument 4-33
`ss_link=<value>` argument 4-33
stamp tool
definition of Glossary-5
standard component
definition of Glossary-5
standard install
definition of Glossary-5
standard processing of shippable files 5-8
`stdout` 4-7
sun4c platform (Solaris) 3-6
syntax rules for index files 3-1 to 3-4
system environment

for mkarea 4-20
for mkmanf 4-16
for mkmm 4-8
for mkmmconv 4-3
for mkmmdeps 4-5

T

tape log file
definition of Glossary-5
tape+<dev>argument
(retired)HyperHot> 4-26
tape=<dev> argument 4-29
tapes
CD-ROM virtual 4-29, 4-30
virtual (Vtape) 4-23, 4-29
tar archive format 3-1, 4-7
tar=<dir> argument 4-22
temporary directory <dir> A-8
text editor 6-3
tmp=<dir> argument 4-22
too few fields A-4
too many fields A-4
toolkit
definition of Glossary-5
overview 1-2
types 3-9
trademarks i
trailing slashes 3-39
tutorial 6-1 to 6-12
typographical conventions xiv

U

uid=root argument 3-18, 3-20, 3-21
uid=root flag 3-15
undocumented custom script functions 5-24
UNIX 5-22
update_app_defs function 5-24
updating
MMML files 4-18
PIB files 5-15
updating PIB files 5-30
upgrading an existing index file A-2
user intervention 5-18

V

VAR (Value Added Reseller) 2-3
variables
${CS_XAPPL_ICONBLINK} 5-27
${DS} 5-27
${SG_ROOT} 5-27
${SG_SUP_ROOT} 5-28
${VAR} 5-27
${WINDOWING_SYS_LIB} 5-27
$AUTO 5-20
$AWKUTIL 5-21
$CPUTYPE 5-20, 5-21
$DEV_NULL 5-18, 5-22
$DEVAREA 6-2
$IROOT 5-22
$LD_LIBRARY_PATH 5-22
$LOG 5-25
$LOGDIR 5-25
$MANF 4-16, 4-23, 6-11
$PATH 5-22
$RANLIB 5-22
$SG_ROOT 5-23
$SG_SUP_ROOT 5-23
$SG_TOOLS_ROOT 5-23
$SS_ROOT 5-23
$SS_TOOLS_ROOT 5-23
$WINDOWING_SYS_BIN 5-25
$WINDOWING_SYS_LIB 5-25
$WINDOWING_SYS_LIB_APP 5-25
$WINDOWING_SYS_TKLIB 5-25
forms–$VAR, ${VAR}, or $(VAR) 3-3
LD_LIBRARY_PATH 5-22
with mkmanf 4-16
with mkmm 4-8
with mkmmdeps 3-6, 4-5
vend: label 3-10, 3-25
verbose option (retired) 4-28
verify=y/n argument (retired) 4-28
version number (SPECTRUM) 1-1
vi 6-3
virtual tape (vtape) 4-23, 4-29
definition of Glossary-5
Virus Disclaimer i
vtape <dir> option (retired) 4-28
vtape=<dir> argument 4-23, 4-26
vtapes 4-29
(virtual tapes) 4-29
CD_ROM 4-29
shipping on magnetic tape 6-8

W

warning messages 4-34
warnings vs. errors A-1
whitespace 3-2
Windows NT 4-1, 4-5, 4-31, 5-19, 5-22
writing custom install scripts 5-16 to 5-19

X

x option (retired) 4-28
X resource file templates 5-26
X resource files
    installing 5-15
X-based windowing system 5-26
xtn option (retired) 4-28
xtn_install=<value> argument 4-33